Leakage Resilient Symmetric Cryptography: Theory and Practice

DANIEL P. MARTIN

A dissertation submitted to the University of Bristol in accordance with the requirements for award of the degree of Doctor of Philosophy in the Faculty of Engineering.

Merchant Venturers School of Engineering

August 2016

c. 74413 words
Abstract

Side channel analysis may be used to break the security of a cryptographic scheme by utilizing information revealed by the physical instantiation of the algorithm (leakage), instead of an inherent weakness in the algorithm itself. Several methods have been proposed to protect a device against leakage but without a proof of security, there is no guarantees that they are secure. Recently theoreticians have tried to address this by constructing schemes which are provably secure against (certain) side channels. Unfortunately these schemes tend to be inefficient.

In this thesis we try to address this balance. We create multiple schemes which are provably secure against a wide range of leakage while still being efficient. To create these schemes we review the security models that support leakage and choose the two most suitable. After showing difficulties with the simulatable leakage model, we focus on constructing schemes in the continuous leakage model.

We conclude by trying to practically capture a theoretical property required by the majority of the theoretical leakage models. Most of the models require that the device only leaks a bounded number of bits (where the bound is less than the key size). However, in practice when measuring the power of a device, the resulting traces can be several megabytes in size, considerably larger than the key size. By constructing the first exact key rank algorithm, we are able to investigate how much information can be extracted from a power trace. The key rank problem states, given the output of a side channel attack, how many keys are more likely than the given key. Using this method we are able to compare the leakage of a device to the theoretical bounds given for certain schemes in the literature. We show that the theoretical bounds closely match how a device behaves in practice.
Acknowledgements

First and foremost my thanks goes to my two supervisors Professor Elisabeth Oswald and Dr Martijn Stam, for their support and guidance throughout my time here at the University of Bristol. I would also like to thank the cryptography group as a whole, with special mention to my co-authors, and Guy Barwell for acting as a sounding board, and proof-reader, whenever needed.

Finally I would like to thank my family and friends for their support. To my mother and father for encouraging me from a young age to fulfil my academic potential, without you I wouldn’t be here today. To my friends, especially Dom and Hannah, for their support during my research. To Cass for her unwaivering support and encouragement during the writing of this thesis.
Declaration

I declare that the work in this dissertation was carried out in accordance with the requirements of the University’s Regulations and Code of Practice for Research Degree Programmes and that it has not been submitted for any other academic award. Except where indicated by specific reference in the text, the work is the candidate’s own work. Work done in collaboration with, or with the assistance of, others, is indicated as such. Any views expressed in the dissertation are those of the author.

SIGNED: .................................................................

DATE: .................................................................
Contents

List of Tables 14
List of Figures 16
Acronyms 23

1. Introduction 25
   1.1. Side Channel Attacks .................................................. 26
       1.1.1. Differential Power Analysis (DPA) ............................... 27
       1.1.2. Simple Power Analysis (SPA) ..................................... 29
       1.1.3. Side Channel Countermeasures .................................... 30
   1.2. In This Thesis ............................................................ 33

2. Preliminaries 37
   2.1. Groups ................................................................. 37
       2.1.1. Elliptic Curves .................................................. 39
   2.2. Assumptions ........................................................... 44
   2.3. Security Notions ...................................................... 47
       2.3.1. Pseudo Random Functions and Generators ...................... 48
       2.3.2. Encryption ..................................................... 51
       2.3.3. Message Authentication Codes (MACs) ......................... 62
   2.4. Models ................................................................. 64
       2.4.1. Standard Model .................................................. 65
       2.4.2. Generic Group Model (GGM) .................................... 65
2.4.3. Random Oracle Model ................................. 67

3. Previous Work .............................................. 69

3.1. Simulatable Leakage ..................................... 70
3.2. Private Circuits ......................................... 71
3.3. Bounded Storage Model ................................. 74
3.4. Bounded Retrieval Model ............................... 75
3.5. Bounded Leakage Model ................................. 77
3.6. Continuous Leakage Model ............................. 78
   3.6.1. Leak Free Components ............................ 79
   3.6.2. Only Computation Leaks Information (OCLI) ...... 80
3.7. Auxiliary Input .......................................... 82
3.8. Relationship Between Models ........................ 83
   3.8.1. Bounded Retrieval Model and Bounded Leakage Model 84
   3.8.2. Continuous Leakage Model and Bounded Leakage Model 84
   3.8.3. Probing Model and the Noisy Leakage Model .......... 84
3.9. Variations of the Bounded and Continuous Leakage Models .......... 85
   3.9.1. Bounded Leakage Functions ...................... 85
   3.9.2. Entropy Preserving Leakage Functions .............. 85
   3.9.3. Simulatable Leakage Functions .................... 86
   3.9.4. Hard to Invert Leakage Functions .................. 87
   3.9.5. Relationship Between the Various Leakage Function Classes .... 88
   3.9.6. Non-Adaptive Leakage Functions ................... 89
3.10. Conclusion ............................................. 89

4. Simulatable Leakage Model ................................ 93

4.1. Previous Work in the Simulatable Leakage Model ............ 94
   4.1.1. The Model ......................................... 94
   4.1.2. The Simulator ..................................... 98
   4.1.3. The Construction .................................. 99
4.2. The New Simulatable Leakage Model ........................................ 101
4.3. Breaking the Simulator and the Challenges of Fixing it ............... 105
   4.3.1. Properties of Real World Leakage .................................... 105
   4.3.2. Cross-Correlation as a Distinguisher ............................... 106
   4.3.3. Distinguishing Between the Simulator and Real Leakage .......... 109
   4.3.4. Experiments on Real Devices ........................................ 110
   4.3.5. Trying to Fix the Simulator ...................................... 112
4.4. Creating a Theoretical Simulator ....................................... 112
4.5. Further Work ............................................................. 114
   4.5.1. Alternate Simulators .............................................. 115
   4.5.2. Schemes Built Upon Simulators .................................. 116
4.6. Conclusions .............................................................. 117

5. Continuous Leakage Model ................................................. 119
   5.1. The Leakage Resilient Security Model ................................ 120
      5.1.1. Leakage and the Generic Group Model ........................... 122
   5.2. Previous Work in the Continuous Leakage Model .................... 123
      5.2.1. Kiltz and Pietrzak Key Encapsulation Mechanism (KEM) [101] . . . 124
      5.2.2. Galindo and Vivek Signature Scheme [75] ...................... 127
      5.2.3. Other Signature Schemes ...................................... 129
   5.3. A Key Update Mechanism .............................................. 129
   5.4. A Leakage Resilient Message Authentication Code .................. 141
      5.4.1. A MAC Without Leakage ....................................... 141
      5.4.2. Model .......................................................... 145
      5.4.3. A Leakage Resilient MAC ..................................... 147
      5.4.4. An Alternative MAC .......................................... 153
   5.5. A Leakage Resilient Encryption Scheme ............................. 160
      5.5.1. A Leakage Resilient Pseudo Random Function .................. 162
      5.5.2. Leakage Resilient Encryption Notions .......................... 176
      5.5.3. A Leakage Resilient Encryption Scheme ........................ 181
5.6. Practical Considerations ................................................. 198
  5.6.1. Security Considerations .................................. 198
  5.6.2. Comparison with Other Leakage Resilient Schemes .... 199
  5.6.3. Random Point Generation .................................. 205
  5.6.4. The Jacobi Symbol .......................................... 209
  5.6.5. Hash Functions ............................................. 212
5.7. Conclusion .......................................................... 212

6. Practically Estimating $\lambda$ Bits of Leakage 215
  6.1. Notation and Preliminary Definitions ......................... 217
  6.2. Previous Work .................................................. 219
    6.2.1. Key Rank .................................................. 219
    6.2.2. Key Enumeration ........................................ 221
    6.2.3. Reflecting on the Previous Work ....................... 222
  6.3. Key Rank .......................................................... 223
    6.3.1. Casting as a Knapsack Problem ......................... 223
    6.3.2. Key Rank Graph ........................................ 228
    6.3.3. Counting Valid Paths .................................. 231
  6.4. Key Enumeration .................................................. 236
    6.4.1. Time Complexity ....................................... 237
    6.4.2. Memory Efficiency ..................................... 240
    6.4.3. Parallelisation ......................................... 243
  6.5. Implementation .................................................. 245
    6.5.1. Precision ................................................ 246
    6.5.2. Factors Influencing the Precision Levels in DPA .... 246
    6.5.3. Experimentally Measuring the Precision .............. 247
    6.5.4. Key Rank ................................................ 250
    6.5.5. Key Enumeration ....................................... 251
  6.6. Relating Key Rank and Leakage Resilience .................... 253
    6.6.1. Theoretical Bounding of Leakage ....................... 254
6.6.2. Practical Bounding of Leakage ........................................... 256
6.6.3. Comparing Both Leakage Bounds ........................................ 258
6.7. Conclusion ................................................................. 259

7. Conclusion ........................................................................... 263
7.1. Publications ......................................................................... 267
7.1.1. In This Thesis ................................................................. 267
7.1.2. Other Publications .......................................................... 271

Bibliography ............................................................................. 273

A. AES ....................................................................................... 297
A.1. Algorithm ........................................................................... 297
A.1.1. KeyExpansion ............................................................... 297
A.1.2. AddRoundKey ............................................................. 298
A.1.3. SubBytes ................................................................. 298
A.1.4. ShiftRows ................................................................. 299
A.1.5. MixColumns .............................................................. 300

B. Generic Group Model .......................................................... 301
B.1. Generic Group Model Oracles ............................................. 301
List of Tables

5.1. A comparison of possible authentication schemes ........................................... 203
5.2. A comparison of possible leakage resilient encryption schemes ....................... 205
5.3. The algorithms for the four random point generation algorithms considered .... 206
6.1. Pseudo code for generating the index of each child of a vertex ....................... 238
A.1. The AES SBox lookup table. The column is determined by the least significant
     nibble and the row by the most significant nibble ........................................ 299
## List of Figures

2.1. Two example elliptic curves over the real numbers .......................... 40
2.2. Two example elliptic curves over $\mathbb{F}_5$ ................................. 41
2.3. A graphical representation of the addition of two elliptic curve points .......................... 41
2.4. PRF experiment .............................................................................. 49
2.5. wPRF experiment ........................................................................... 49
2.6. PRG experiment .............................................................................. 51
2.7. IND-CPA experiment ...................................................................... 53
2.8. IND-CCA1 experiment .................................................................... 54
2.9. IND-CCA2 experiment .................................................................... 54
2.10. RoR-CPA experiment ..................................................................... 56
2.11. RoR-CCA experiment ..................................................................... 56
2.12. IND$^+$-CPA experiment ................................................................. 57
2.13. IND$^+$-CCA experiment ................................................................. 58
2.14. nAE experiment ............................................................................. 59
2.15. OW-CPA experiment ...................................................................... 60
2.16. OW-CCA experiment ...................................................................... 60
2.17. INT-PTXT experiment ................................................................... 61
2.18. INT-CTXT experiment ................................................................... 62
2.19. EUF-CMA experiment .................................................................... 63
2.20. sEUF-CMA experiment .................................................................. 64

3.1. A Leakage Resilient PRG [133] ....................................................... 81
3.2. Set relations for different classes of leakage functions ...................... 87
4.1. \( q \)-simulatable leakage ................................................. 96
4.2. Description of simulator \( S_{\text{sec}}^l \) [158] ................................ 98
4.3. Left: Leakage Resilient PRG \( G \). Right: 2PRG construction ............. 100
4.4. A visual depiction of the adversary interaction with the \( \text{Enc} \) oracle \( b = 0 \) in the \( q \)-sim game ................................................................. 101
4.5. A visual depiction of the adversary interaction with the \( \text{Enc} \) oracle \( b = 0 \) in the \( p-q \)-sim game ................................................................. 102
4.6. \( p-q \)-simulatable leakage .................................................... 103
4.7. Power traces for an 8051 8-bit microcontroller [108] ....................... 105
4.8. AES power traces and cross-correlation plots [108] .......................... 109
4.10. The adjusted PRG construction [108] ..................................... 113
4.11. A generic simulator \( S \) secure against the cross-correlation distinguisher [108]. 113
4.12. Description of simulator \( S_{\text{secure}} \) [132] ................................. 115

5.1. A visualisation of the splitting and leaking upon a function ............... 121
5.2. KP KEM K [101] ................................................................. 124
5.3. KP Leakage Resilient KEM \( K^* \) [101] .................................... 125
5.4. GV Signature scheme \( S \) [75] .................................................. 127
5.5. GV Leakage Resilient Signature scheme \( S^* \) [75] ......................... 128
5.6. A visualisation of a key update mechanism .................................. 130
5.7. KP Key Update Mechanism \( KU \) ........................................... 131
5.8. The algorithm \( F^* \) ............................................................... 133
5.9. \( L \)-CKR-UL experiment ..................................................... 134
5.10. \( L \)-DKR-UL experiment ...................................................... 135
5.11. An adversary against the \( L_{\text{update}} \)-DKR-UL security of \( KU \) ........ 140
5.12. Constructing a CBDH Adversary from a TBDH Adversary ............... 142
5.13. Our bilinear MAC scheme \( M \) .............................................. 144
5.14. Constructing a TBDHwO Adversary from a sEUF-CMA Adversary .... 144
5.15. \( (L_{\text{tag}}, L_{\text{vrfy}}) \)-sEUF-CMLA experiment ............................. 146
5.16. Leakage Resilient MAC $M^*$ ........................................... 147
5.17. Our alternate bilinear MAC scheme $M_1$ .............................. 154
5.18. Constructing a EUF-CMA Adversary Against $M$ from a EUF-CMA Adversary Against $M_1$ .................................................. 155
5.19. Alternate Leakage Resilient MAC $M_1^*$ ............................... 155
5.20. Explicit PRF Definition ..................................................... 163
5.21. $L$-PRLF experiment ....................................................... 165
5.22. The PRLF $F^*$ ............................................................. 166
5.23. The underlying PRF $F$ ....................................................... 167
5.24. Adversary $B$ simulates the hybrid games $H_{i-1}, H_i$ for adversary $A$ . 168
5.25. An adversary against $F$ when split into two shares ($L_{prf}$-PRLF) . 169
5.26. nALE experiment .......................................................... 177
5.27. $\mathcal{L}$-IND$^\$-CPLA experiment ...................................... 178
5.28. aIND$^\$-CPLA experiment .............................................. 179
5.29. CFB Mode; Encryption (Left) and Decryption (Right) ............... 182
5.30. The game $G_1$, used to prove IND$^\$-CPLA security of CFB mode .... 183
5.31. The game $G_2$, used to prove IND$^\$-CPLA security of CFB mode .... 184
5.32. The game $G_3$, used to prove IND$^\$-CPLA security of CFB mode .... 184
5.33. A PRLF adversary $B$ constructed using a distinguisher $A$ for $G_1, G_2$ .......... 185
5.34. A5 ............................................................................. 188
5.35. The game $G_{a_5}$, used to prove nALE security of A5 .................. 193
5.36. The game $G_{b_5}$, used to prove nALE security of A5 .................. 194
5.37. The game $G_{c_5}$, used to prove nALE security of A5 .................. 195
5.38. The game $G_{d_5}$, used to prove nALE security of A5 .................. 196
5.39. The game $G_{e_5}$, used to prove nALE security of A5 .................. 197
5.40. EM traces for an RSA decryption on a Beagle Bone Black. ................ 207
5.41. Power traces for the Jacobi symbol [115] ............................... 211
6.1. Example probabilities for \( m \) key chunks. Each chunk can take values from 0 to \( n - 1 \). The values that correspond to the (hypothetical) secret key are highlighted in grey.

6.2. The construction of the graph for the small example instance provided [113].

6.3. The Key Rank Algorithm.

6.4. The array generated when running the key rank algorithm for the running example.

6.5. The key enumeration algorithm.

6.6. Different options for topological sorting for our previous example. Note that the deepest vertex in each chunk will be guaranteed to have a left child leading to \( R \); for clarity these paths are omitted [113].

6.7. The key tree for all possible three character keys containing ‘\( \alpha \)’ or ‘\( \beta \)’.

6.8. Bits of precision for Key Rank (left) and number of bins for Glowacz et al. [78] (right) [113].

6.9. Impact of the distinguisher (left: correlation, right: Gaussian templates without \( \log_2 \)) on the precision requirements [113].

6.10. Observed difference in calculated key rank between our algorithm and Glowacz et al. [113].

6.11. Impact on the size of \( W \) against the runtime [113].

6.12. Impact on the precision kept against the runtime [113].

6.13. Comparison of the runtime of our algorithm and Veyrat-Charvillon et al.’s algorithm (single core) [113].

6.14. Comparison of the runtime on a varying number of cores [113].

6.15. The Leakage Resilient PRG [133].

6.16. Plotting the theoretical leakage bounds.

6.17. \( \lambda \) leakage compared to the number of traces.

6.18. Comparison of the theoretical and observed leakage bounds.

7.1. An alternative VRFY for the MAC \( M^* \).
B.1. GGM group oracles used within various proofs . . . . . . . . . . . . . . . . . 301
Acronyms

**BN** Barreto-Naehrig.

**CAESAR** Competition for Authenticated Encryption: Security, Applicability, and Robustness.

**CBC** Ciphertext Block Chaining.

**CFB** Ciphertext Feed Back.

**CTR** Counter.

**DAG** Directed Acyclic Graph.

**DPA** Differential Power Analysis.

**EEA** Extended Euclidean Algorithm.

**EM** Electromagnetic Radiation.

**FPTAS** Fully Polynomial Time Approximation Scheme.

**GBGM** Generic Bilinear Group Model.

**GGM** Generic Group Model.

**KEM** Key Encapsulation Mechanism.

**MAC** Message Authentication Code.

**MPC** Multi Party Computation.
**NP** Non-deterministic Polynomial-Time.

**OCLI** Only Computation Leaks Information.

**OFB** Output Feed Back.

**PRF** Pseudo Random Function.

**PRG** Pseudo Random Generator.

**PRLF** Pseudo Random Leaky Function.

**RAE** Robust Authenticated Encryption.

**RUP** Release of Unverified Plaintexts.

**SGX** Software Guard Extension.

**SIV** Synthetic IV.

**SPA** Simple Power Analysis.

**SSL** Secure Sockets Layer.

**wPRF** weak Pseudo Random Function.
1. Introduction

In a world where a vast amount of information is easily accessible, on a variety of portable electronic devices, cryptography has become increasingly important. This is because nowadays many users are concerned about their privacy. Cryptography can help to alleviate privacy concerns by offering ways to achieve data (and entity) authentication, data confidentiality, *etc.*

It is now folklore that Caesar used a cipher to communicate with his generals. This was later broken using statistical analysis on the frequency of the letters in the ciphertext. It can now sometimes be found in the puzzle section of a newspaper. For many years cryptography was in this exact situation, a cat and mouse like game in which someone would design a scheme which would be used until someone else found a way to break it. From around the 1980s [81] this design paradigm for cryptography changed and the community started designing schemes which were provably secure. A scheme is shown to be secure providing an underlying problem is hard. A problem is hard to solve if it is computationally expensive to calculate the answer (there is no known polynomial time algorithm). The problem chosen tends to have been studied for tens (or hundreds) of years and therefore is generally believed to be hard. An example problem is integer factorisation. This resulted in secure cryptographic algorithms, for as long as the underlying problem remained hard to solve.

In 1996 Kocher (*et al.*) [102, 103] showed how a, previously thought to be secure, algorithm could be attacked using side channels. A side channel is a source of information which arises from the implementation of an algorithm on a specific piece of hardware, instead of from the algorithm itself. Commonly exploited side channels include timing, power consumption and Electromagnetic Radiation (EM). Side channels have enabled schemes with a proof of security to be broken. This is because the security model in which the scheme is proven secure does not
capture side channels. The discovery of side channels seems to have reverted the mind set of cryptographers back to the cat and mouse style of design. While schemes are still designed with a proof, the back and forth has moved to the side channel countermeasures, a method to stop the side channel being exploited. Someone will come up with a countermeasure to stop side channel attacks which will be used until a new side channel attack is devised which circumvents the new countermeasure. At this stage the whole cycle repeats. Side channels and the common countermeasures will be discussed in Sect. 1.1. While they are not the focus of this work they are an important step in the evolution of the topic that will be under discussion. The efficiency of any new scheme is compared to old schemes which implement countermeasures and therefore an understanding of these common countermeasures is important.

In recent years cryptographers have tried to bring the rigour of provable security to the accompanying countermeasures. The goal is to create new theoretical models which capture either some, or all, side channels. Therefore when a scheme is proven secure in this new model, when implemented, it will be secure against those particular side channels. One challenge is constructing a model which secures algorithms against side channels that have not been discovered yet. It may be possible to construct schemes secure against such a range of attacks that the scheme is secure before a new side channel is discovered. Another challenge is how to model the leakage assumptions. If the assumptions are too restrictive they will not allow all side channels to be captured, while if they are too general it may be hard to construct efficient schemes in the resulting model. It is this area of cryptography in which this thesis will focus. The goal of this work is to choose a suitable leakage model to construct schemes which are secure against as many side channel attacks as possible (and ideally those still to be discovered) while being competitive with the countermeasures that are used without proof.

1.1. Side Channel Attacks

In theoretical cryptography, schemes are proven secure in security models similar to those found in Sect. 2.3 by reducing their security to hardness assumptions, such as those in Sect. 2.2. Reducing the security of a scheme to a hardness assumption, implies if the scheme can be shown
insecure then the assumption can be shown to be false. Therefore if the assumption holds, the scheme is secure. However, when schemes are implemented, even when they are implemented correctly, they are sometimes not secure. This tends to be through the use of side channel attacks (e.g. timing, power or EM) which enable the extraction of the secret data, as initially detailed by Kocher (et al.) [102, 103]. These attacks do not correspond to ‘breaks’ of the scheme in its security model nor do they correspond to falsifying the underlying hardness assumption. These breaks arise because getting the side channel information on the secret is not captured by the security model.

In the remainder of this section two types of attacks are discussed involving using power consumption as the side channel. The main difference between the two attacks is how the data within the trace is utilised (DPA exploits data dependent features which only become apparent across many inputs, while Simple Power Analysis (SPA) exploits features that are apparent from few inputs). There are two methods for exploiting side channel information; horizontal attacks and vertical attacks [16]. We will focus on vertical attacks where each trace/measurement corresponds to a single (complete) execution of the algorithm on a given input [110]. Therefore, the adversary will be exploiting the same information within a trace across many inputs. In horizontal attacks each measurement corresponds to a single ‘period’ of the same execution [165]. Therefore, the adversary will be exploiting different points from a single input (or a few inputs). The section is concluded with a discussion of countermeasures to protect against side channel attacks. The goal is to stop side channel attacks, so that the instantiation of the scheme is brought closer to the security model in which it was shown to be secure.

1.1.1. Differential Power Analysis (DPA)

Differential Power Analysis (DPA) tends to be the preferred method of attacking a piece of cryptographic hardware [110], to extract the secret key. DPA is the favoured method of attacking a device, as it does not require a detailed knowledge of how the device is performing its computation but just what it is performing. The key is normally retrieved key chunk at a time, where a key chunk is defined as a (normally independent) part of the key. Taking AES as an example (it will become a running example in this section. See App. A for details), only high level knowl-
edge of the implementation is required. Such as, if the implementation uses an SBox look-up or a T-table. In AES the key chunks tend to be the independent bytes of the key but sometimes 32 bits of the key are considered instead. The latter case represents working on a 32 bit architecture while the former case represents a simpler 8 bit architecture. DPA (normally) requires a large number of traces compared to other methods, but can handle noise within the traces better than other methods. Increasing the noise within a trace can be countered by increasing the number of traces, to get the same quality results.\footnote{Hence, if the traces are reasonably noise free, it may be possible to perform a DPA attack with reasonably few traces.}

DPA works by exploiting the data dependency within a trace. The traces will be analysed at a fixed moment in time as a function of the input data (the secret key will be fixed for all traces). The method used to perform this analysis is described below.

The first stage of a DPA attack is to choose which intermediate value to attack (which fixed moment in time). The requirement is that it is at a point in the function which can be written as \( f(K, M) \) where \( f \) is an efficiently computable function, \( K \) is the secret data to be learnt (normally a key chunk) and \( M \) is an input which can be controlled. In AES \( f(K, M) = SBox(K \oplus M) \) can be used, meaning the attack targets just after the (first round) SBox has been applied and will be trying to learn a single byte of the key (the attack can be repeated 16 times to recover the whole key). The next stage is to collect power traces from the device that is being attacked such that each trace has a different (known) value for \( M \) in the intermediate function. When the power traces have been collected, for each value of \( M \) used a hypothetical intermediate value \( F_i = f(i, M) \) should be computed for all possible values of the key chunk \( i \). This step shows why it is important to divide the key into parts to attack because it is not feasible to run this step for all \( 2^{128} \) keys but working on key bytes it is possible to do it for all 256 candidates (if using 8 bit key chunks). The size of the key chunks chosen determines the amount of enumeration effort required at this stage of the DPA attack. The hypothetical intermediate values are then mapped to hypothetical power consumption values using a leakage model. One of the standard leakage models is Hamming weight leakage \([37, 110]\), where a binary value’s Hamming weight is defined as the number of bits set to 1. Given these hypothetical power values, they are compared to the real traces and the value of \( i \) which leads to the ‘most similar’
hypothetical power values is assigned to be the key byte. The normal metric used for similarity is correlation, where the vector of hypothetical power values (for each key chunk) is compared to each position within the power traces and the score for this key is then the, absolute, maximum of all these results. The key returned is then simply the key value with the highest correlation score.

Sometimes (e.g. when the traces are particularly noisy), the key may not be the most likely as returned from the DPA attack and multiple key candidates may need to be tested. In this situation key enumeration can be used to test the keys in order of most likely first and this will be discussed further in Ch. 6.

The technique above is described for power traces, but can just as easily be applied to EM traces.

Schemes which are provably secure in the face of leakage must protect against DPA style attacks. The standard approach to protect against DPA attacks, and the one used within this work, is to frequently update the key. Updating the key in a provably secure manner is discussed in Sect. 5.3. By frequently updating the key, the adversary might be unable to get enough traces on a key to perform a successful DPA attack. Without multiple traces on the same underlying key, an adversary must try to extract the key with few traces. In this situation SPA attacks may be the best method for an adversary to extract the key, and will be discussed next.

1.1.2. Simple Power Analysis (SPA)

In contrast to DPA, SPA utilises either one or very few inputs to try to extract the secret key.\(^2\) It does this by trying to exploit key dependent differences within a trace, compared to DPA which exploited data dependent differences across multiple traces [110]. The advantage of this methodology is that it uses very few traces and more leakage can be utilised, compared to DPA which normally requires a lot of traces and tends to only exploit very few points per trace. The disadvantage is that any noise within the trace tends to be extremely detrimental to the results. There has been a whole host of work producing SPA attacks on AES and other schemes [102, 109, 149, 150]. As an example, Mangard [109] uses leakage from the AES key expansion to

\[^{2}\text{It may be possible for an adversary to take multiple traces per input to help reduce the noise.}\]
reduce the key space such that brute force over the remaining keys is possible.

Schemes which are provably secure in the face of leakage must also protect against SPA attacks. Some SPA attacks are prevented by the model, such as the bounded leakage model which restricts the number of bits learnt per trace to \( \lambda \) bits. Leakage models are discussed in Ch. 3. Without these restrictions an adversary could just ask for the secret. Schemes prevent SPA attacks using side channel countermeasures. Countermeasures are discussed in the next section. To construct schemes provably secure in the face of leakage, countermeasures that come with a proof of security must be used.

1.1.3. Side Channel Countermeasures

In this section we introduce countermeasures against side channel attacks. The three we focus on are masking, secret sharing and blinding. All three methods share similarities in how they protect the underlying secret information. Each method achieves protection by not directly computing on the secret but computing on a related representation. The unpredictability of the representation, makes it more complex for an attacker to exploit the side channel leakage. It is the different representations that differentiates between the countermeasures. While the three methods share similarities with how they achieve protection against side channel attacks, we have chosen to separate the three due to slight differences between them. Masking can be seen as the most general of the three, and is typically used for protection of block ciphers. Secret sharing tends to come with a proof of security (recently masking has also come with security proofs [136]). It is normally considered in the case where each party is given distinct share of the secret, but is easy to view where all shares are owned by the same party and operated disjointly in time and space (e.g. not using the same register to avoid joint leakage). Blinding is similar to masking except that it exploits some underlying structure, such as a group, to provide security.

**Masking**

In masking, a mask \( m \) is used to mask a (secret) value \( v \) by computing \( v_m = v \star m \) where \( \star \) tends to correspond to an operation of the underlying scheme which is being masked. For example, exclusive-or can be used on schemes defined over bitstrings, while addition can be used within
a finite field. The new version of the scheme will then work with the mask values $v_m$ and $m$ instead of the underlying values and will produce a result that is also masked and thus the masks must be removed at the end of the scheme. All intermediate values (involving the secret) must be masked otherwise it could be possible to attack the unmasked portion.

Using a masking scheme will protect a device against a DPA attack and therefore a new attack strategy had to be defined. This is known as a second order DPA attack [110]. In response to this, the number of masks used was increased to protect against this form of attack. This has led to a direct relation between the number of masks used and the order of the DPA attack performed to extract the secret from the device. If masking is implemented, the standard choice is to use masking with a single mask, despite the risk of higher order DPA attacks. The reason for this is that every extra mask added greatly reduces the efficiency. Since these devices are required to be as efficient as possible and extra masks only increase the adversary’s attack complexity by a known amount, a single mask was chosen as the best balance of efficiency and security.

Recently there have been several masking schemes that come with a proof of security. The goal of these schemes is to create (efficient) masking schemes which are provably secure against DPA attacks of a given order. Prouff and Rivain [136] prove the security of masked implementations of block ciphers under the Only Computation Leaks Information (OCLI) assumption [120] (this assumption will be discussed in more detail in Sect. 3.6.2 and Ch. 5). Prouff and Rivain’s proposed masking scheme is the first work to show how to provably make the entire block cipher secure against DPA. However, it requires a leak free component to refresh the mask. The scheme by Chari et al. [43] came earlier but focuses on when the masks leak independently of any computation. The work by Ishai et al. [92] is similar but focuses on the probing model (this work will be discussed in more detail in Ch. 3).

**Secret Sharing**

Masking can be related to the theoretical approaches of secret sharing [153] and Multi Party Computation (MPC) [24, 79]. Secret sharing takes a secret and splits it into $t$ ‘shares’ such that having less than $t$ of the shares gives an adversary no more information about the secret than if they had access to none of the shares, yet with all $t$ shares they can recover the secret.
A generalisation splits the secret into \( t \) shares such that if an adversary has less than \( n \) shares they learn no more information than if they had no shares, yet with \( n \) (or more) shares they can completely recover the secret. MPC allows a group of people to compute a function, where each person has their own secret input and everyone receives the output such that each party learns no more about the other parties’ secret than the output reveals. A MPC scheme tends to be built on top of a secret sharing scheme due to the similarity of the problems. Using a general MPC scheme it is possible to construct a masking scheme [47]. However, in general, designing a dedicated masking scheme tends to be more efficient. Belaïd et al. [18] discuss whether masking or provable security against leakage should be used. They conclude that when the amount of leakage can be bounded (such as in a Pseudo Random Generator (PRG) due to its stateful nature) then the provable security is more desirable, yet while the leakage is harder to bound (such as for a Pseudo Random Function (PRF) which is stateless) masking is a more desirable solution. Interestingly, and somewhat counter-intuitively, they show that it is not worth combining both forms of protection in the particular cases studied.

In Ch. 5 we combine a secret sharing/masking scheme with the OCLI assumption (defined in Sect 3.6.2), to produce a host of leakage resilient cryptographic primitives.

**Blinding**

Blinding protects a scheme against DPA style attacks by using a different representation of the secret value per function call [110]. The representation must be chosen such that it computes the same output as using the original secret. Using a different representation per call stops an adversary getting repeated leakage on the underlying secret value. It is not enough to repeatedly use a single representation which computes the same results as the secret because then DPA could be used to learn this value and since it behaves exactly the same as the secret, it is just as valuable to the adversary and hence a different value must be used each time. Blinding tends to be used in asymmetric cryptography because it requires some form of structure (such as a group) to calculate the representations to use (blinding can’t be done for something like AES). As an example, in an elliptic curve, since exponentiation is calculated modulo \( p \) (for a suitable field choice), instead of using the exponent \( c \), the value \( c + d \cdot (p - 1) \) will give the same result for
any integer $d$ but since the value used is different it makes attacking the scheme more complex. Another example (and one used frequently in the elliptic curve literature [70,73]) is when testing if a value $x$ corresponds to a point on an elliptic curve, in particular if $x^3 + a \cdot x + b$ is square. Instead of testing $x$ directly, a random value $r$ can be chosen and $r^2 \cdot (x^3 + a \cdot x + b)$ is tested to determine if it is square. It is reasonably easy to see that $r^2 \cdot (x^3 + a \cdot x + b)$ is square if, and only if, $x^3 + a \cdot x + b$ is also square and thus both computations return the same result.

In Sect. 5.6 we discuss how blinding can be used to reduce leakage which can be exploited by an SPA style attack against our leakage resilient constructions.

Other Countermeasures

There are a host of other countermeasures that can be applied to a device, including shuffling and hiding [110]. Shuffling involves performing, independent, parts of the computation in a random order each time the function is called. Shuffling makes it hard for an adversary to determine what part of the power trace corresponds to which part of the computation. Unlike the previous countermeasures, hiding does not change the computations performed by algorithm. Hiding countermeasures try to make the power consumption of a device look ‘as random or uniform as possible’ to try and stop it being exploitable by an adversary. One attempt to try and achieve this is Dual Rail Logic [143]. Dual Rail Logic hides the power consumption by transmitting the value over a pair of wires instead of a single wire. One of the wires carries the original value while the other wire carries the negation of this value. Therefore all pairs of wires will contain a one and a zero, thus making it harder to exploit the leakage as the Hamming weight is always half of the total number of wires. Since these countermeasures do not come with a proof of security they will not be discussed for the remainder of this thesis.

1.2. In This Thesis

After discussing the mathematical preliminaries, Ch. 2 introduces the common security models that are used by the cryptographic community to prove the security of a scheme when leakage is not involved. These definitions are important for two reasons. Firstly security with leakage

26
cannot be achieved if there is no security without leakage and therefore all schemes given within this work must also be shown to be secure under these definitions. Secondly, the definitions involving leakage will build on the definitions without leakage and thus an understanding of the original definitions is important. The chapter then gives the assumptions that security will be reduced to throughout this work. The chapter concludes by discussing the different models that security will be proved within during the course of this work.

In Ch. 3 a survey is performed of all the prevalent security models that try and capture a general class of leakage. While the countermeasures discussed previously aim to protect against a particular side channel attack, the goal of the models surveyed is to capture any side channel attacks that meet a particular set of requirements. It is these requirements and restrictions that form the main difference between all of the leakage models.

Chapter 4 explores the simulatable leakage model in more detail. It is a fairly new model that captures leakage and has some desirable properties to it. Thus, we decided to investigate the model in more detail. We start by refining the model such that it more closely represents how both a device and an attacker behave in the real world. We go on to show that the simulator previously given in the literature can in fact be distinguished from real leakage. We conclude the chapter by giving a new simulator which no known adversary can distinguish from a real leakage trace. The downside of this simulator is that it is not particularly efficient and therefore cannot be implemented and compared by evaluation labs. However, since there have been several pieces of work which assume the existence of such a simulator, it is important that one is in place.

After the issues with the simulatable leakage model, Ch. 5 considers the continuous leakage model where there is a restriction to when items are allowed to leak (informally a secret is only allowed to leak when it is computed on). In this model there were few constructions, and even fewer efficient constructions. We construct several new, efficient, schemes and are the first to implement any schemes within this model. We construct a Message Authentication Code (MAC), a PRF and an encryption scheme. Many schemes are designed to be secure against leakage but never get implemented to see if they work in practice. We find this counter-intuitive and therefore implement one of our schemes to see how it behaves in practice. This led to a lot of interesting design decisions which we discuss here. We make several design decisions and
explain why they help to reduce the leakage of the device, so that each function call can meet the theoretical leakage bound, allowing the whole scheme to be secure.

One issue with the continuous leakage model is the lack of method to calculate how many bits a device leaks in practice. To capture this we draw on a disjoint area of the side channel literature. Chapter 6 looks at the seemingly unrelated problem of key rank. Key rank states given a list of probabilities for each key chunk of a key, calculate how far down the sorted list of keys the target key is without sorting them all into order. We give the first key rank algorithm which, given enough precision, has the ability to be exact. Not only is it exact but it is still competitive with the previous estimation algorithms. We then show that our algorithm can be converted into a fairly efficient key enumeration algorithm as well. This is the first algorithm that can easily be adjusted to tackle both problems. Using the exact key rank algorithm we are able to rank the key repeatedly as more traces are added. This can be seen as a measure of how much each trace leaks. We compare the measured value to the theoretical leakage bound given on schemes in the continuous leakage model and see that they are extremely similar. This is the first time that a work has tried to practically calculate the leakage bound used within a theoretical leakage paper.
2. Preliminaries

This chapter introduces all of the foundational concepts which are required for the remainder of
the thesis. It begins with the mathematical definitions that will be required (Sect. 2.1). Any new
concepts or assumptions will be introduced before they are used and then linked back to these
definitions. The basic assumptions on the behaviour of cryptographic groups are given next
(Sect. 2.2). These assumptions will be required to prove security of schemes. Again, variations
of these basic assumptions will be given throughout this thesis, as required, and linked back to
the corresponding assumption. Various cryptographic security definitions are then given without
leakage (Sect. 2.3). This allows them to be expanded upon and have leakage introduced in the
relevant sections. While it is possible to reduce the security of a scheme to a problem given on
groups, these problem are only assumed to be hard problems: It remains a, long standing, open
question as to the difficulty of these problems. To conclude this chapter, we give two (non-
standard) models, which can be used to prove some of these assumptions (Sect. 2.4), as well as
to prove the security of other schemes within the thesis.

2.1. Groups

The majority of schemes given in this work will be built upon a mathematical group. In this sec-
tion the basic definition of a group is given before elaborating on this in various ways, including
a concrete instantiation of a group. The definition of a group is given below for completeness
(we use the notation of [94]).

Definition 1 (Group). Given a tuple \( G = (S, \circ) \) where \( S \) is a (non-empty) set and \( \circ \) is a binary
operator, \( G \) is a group if the following four axioms hold:
Closure \( a \odot b \in S \forall a, b \in S \).

Associativity \((a \odot b) \odot c = a \odot (b \odot c) \forall a, b, c \in S\).

Identity \( \exists i \in S: a \odot i = i \odot a = a \forall a \in S \). We will call this element either 0 or 1 depending on the context it is being used.

Inverse \( \forall a \in S, \exists b \in S: a \odot b = b \odot a = i \). \( b \) is called the inverse of \( a \).

From now on groups will either be written additively (+) or multiplicatively (·) unless otherwise stated.

**Definition 2** (Group Order). Given a finite group \( G = (S, \odot) \), the order of the group is defined as the number of elements in the set \( S \).

**Definition 3** (Homomorphism). A map \( \phi : G_2 \to G_1 \), between two groups \( G_1, G_2 \), is called a homomorphism if:

\[
\phi(g \cdot h) = \phi(g) \cdot \phi(h) \forall g, h \in G_2.
\]

**Definition 4** (Cyclic Group). A (multiplicative) group \( G \) is cyclic if there exists some \( g \in G \) such that:

\[
\exists k \in \mathbb{Z} : a = g^k \forall a \in G.
\]

The element \( g \) is called a generator of the group \( G \).

From this point onward we will be using cyclic groups within this work. Cyclic groups are required for building a bilinear map (also known as a pairing) which is defined below. A pairing is a (normally hard to invert) map between groups which provides additional functionality, allowing a host of cryptographic algorithms to be constructed [33, 95, 127].

**Definition 5** (Bilinear Map). Given three cyclic groups \( G_1, G_2, G_3 \) of order \( p \) with generators \( g_1, g_2, g_3 \) respectively, a function \( e : G_1 \times G_2 \to G_3 \) can be defined. The function \( e \) is a bilinear map if the following properties hold:
**Bilinearity** \( e(g_1^u, g_2^v) = e(g_1, g_2)^{u \cdot v} \forall u, v \in \mathbb{Z}_p. \)

**Non-degeneracy** \( e(g_1, g_2) \neq 1. \)

A bilinear map is also referred to as a pairing. Galbraith et al. [72] categorise pairings into three types:

- **Type 1** \( G_1 = G_2. \)
- **Type 2** \( G_1 \neq G_2 \) but there is an efficiently computable homomorphism \( \phi : G_2 \rightarrow G_1. \)
- **Type 3** \( G_1 \neq G_2 \) and there is no efficiently computable homomorphism between \( G_2 \) and \( G_1. \)

Since \( G_1 \) and \( G_2 \) are cyclic groups of order \( p \), there will always exist a homomorphism between the two groups. Let \( g_1, g_2 \) be generators of \( G_1, G_2 \) then \( \phi(g_1^k) = g_2^k \) is a homomorphism between the two groups. However, in certain instances, the discrete logarithm problem (see Def. 6) in \( G_1 \) must be solved to construct the homomorphism. The discrete logarithm is (normally) assumed to be hard. A homomorphism is called efficiently computable if it can be created without solving the discrete logarithm.

From this point onward we will be using type three pairings (unless otherwise stated). Let \( G_1, G_2, G_3 \) be groups, of order \( p \), with generators \( g_1, g_2, g_3 \) respectively with a bilinear map \( e : G_1 \times G_2 \rightarrow G_3 \) between them, such that \( g_3 = e(g_1, g_2). \) We denote \( \mathbb{BG} = (G_1, G_2, G_3, e, p). \)

In the following subsection we give a concrete instantiation of all of the components described above. When any of the schemes provided within this work were implemented, the instantiations used are the ones given. We begin by discussing elliptic curves, as this will be the group used for implementation. It was chosen as it supports a bilinear map. The Barreto-Naehrig (BN) family of curves is then given due to its efficient implementation. We finally give the Optimal Ate pairing due to its efficiency when combined with a BN curve, compared to other pairings (such as Tate, Weil and Ate).

### 2.1.1. Elliptic Curves

While for the majority of this work we will use an abstract group (normally modelled in the Generic Group Model (GGM) - see Sect 2.4.2), when it comes to instantiating the schemes in
hardware, a concrete group will need to be chosen. Elliptic curves are used because it is possible to construct pairings over (certain) elliptic curve groups. Elliptic curves will be described below.

An elliptic curve $E$, defined over a finite field $K$, is given by the following equation:

$$y^2 = x^3 + a \cdot x + b, \text{ where } a, b \in K.$$  

The curve also needs to be non-singular (i.e. it has no cusps, self intersections or isolated points). A curve is non-singular if, and only if, the discriminant $\Delta$ is not equal to zero, where

$$\Delta = -16(4 \cdot a^3 + 27 \cdot b^2).$$

For illustrative purposes, Fig. 2.1 gives two examples of elliptic curves over the real numbers. Both curves are equally valid choices and the style of curve chosen tends to be just personal preference. For comparison, Fig. 2.2 gives the points on the elliptic curve over the finite field $\mathbb{F}_5$. While elliptic curves are a lot easier to visualise and reason about over the real numbers, in practice they will always be instantiated over a finite field.

Let $E(K)$ be the set of coordinates that satisfy the elliptic curve, together with a point at infinity $\mathcal{O}$. Addition can be defined graphically (example in Fig. 2.3): to add point $A$ to point $B$, a straight line is drawn between the two points, and the third point that intersects the line is denoted as $-(A + B)$. To determine $A + B$ the sign of the $y$ coordinate is negated. This is equivalent to mirroring the point along $x = 0$. If the line does not intersect the curve in a
third location, the result is defined to be \( \mathcal{O} \). Since from this point forward we shall be working over prime fields, the formal definition of addition over prime fields is given. It is possible to give similar equations for binary fields. The method of drawing lines between points helps with intuition but when working over finite fields this is no longer possible and thus the formulas must be used.

The addition \((+\rangle\) operation on the set \( E(K) \) is formally defined below (Assumes \( K \) does not have characteristic two or three. Where the characteristic is the smallest integer \( m \) such that \( m \) times the identity gives back the identity). Given two points on the curve \( A = (x_a, y_a), B = (x_b, y_b) \) to calculate \( C = A + B \) (if \( x_a \neq x_b \)):
\[ s = \frac{y_a - y_b}{x_a - x_b}. \]
\[ x_c = s^2 - x_a - x_b. \]
\[ y_c = y_a + s \cdot (x_c - x_a). \]

If \( x_a = x_b \) and \( y_a = -y_b \) then \( C \) is the point at infinity, while if \( y_a = y_b \):

\[ s = \frac{3 \cdot x_a^2 + a}{2 \cdot y_p}. \]
\[ x_c = s^2 - 2 \cdot x_a. \]
\[ x_c = y_a + s \cdot (x_c - x_a). \]

Similar formulas can be given for the binary field. However, they will not be given here as this work focuses on the prime (non-binary) case.

**Proposition 1.** Given an elliptic curve \( E \) over a finite field \( K \), the tuple \( (E(K),+) \) forms a group.

**Barreto-Naehrig (BN) Curves**

When a scheme is implemented a particular curve has to be chosen. A particular favourite choice of curve (and one we shall use for implementation) is a BN curve [14] due to their versatility; they are computationally efficient to generate, can be implemented in an efficient manner for a variety of scenarios and support a pairing. A BN curve is a curve of the form:

\[ y^2 = x^3 + b, \text{ where } b \in \mathbb{F}_p. \]

For a prime \( p \), such that the curve has the following properties. The number of points on the curve \( n = \#E(\mathbb{F}_p) \) is prime (that is to say the curve has prime order) and for some \( u \in \mathbb{Z} \) the primes \( p, n \) can be written as:
\[ p = 36 \cdot u^4 + 36 \cdot u^3 + 24 \cdot u^2 + 6 \cdot u + 1. \]
\[ n = 36 \cdot u^4 + 36 \cdot u^3 + 18 \cdot u^2 + 6 \cdot u + 1. \]

BN curves have an embedding degree \( k = 12 \), which means the pairing will map to the multiplicative group \( \mathbb{F}_{p^k}^* \), and due to a sextic twist (informally, it is a mapping to another curve which is isomorphic under the closure of \( K \)) it is possible to set \( \mathcal{G}_2 = E'(\mathbb{F}_{p^2})[n] \). Note that these groups \( \mathcal{G}_1, \mathcal{G}_2, \mathcal{G}_3 \) are all of different sizes, while the pairing definition has each group being the same size. While the groups are different sizes, the schemes will all work within a subgroup of the correct size (which exists based on the choice of groups). There exists an optimal ate pairing with loop order \( \omega = |6 \cdot u + 2| \), which is discussed below.

**Optimal Ate Pairing**

Due to the compatibility with a BN curve we will use an optimal ate pairing [162] when we require a pairing for implementation and thus discuss it in more detail here. For the remainder of this section we follow the discussion (and notation) of Beuchat et al. [28] on the optimal ate pairing. The optimal ate pairing \( e_{opt} : \mathcal{G}_1 \times \mathcal{G}_2 \to \mathcal{G}_3 \) is defined as:

\[
(P, Q) \mapsto (f_{6\cdot u+2,Q}(P) \cdot l_{[6\cdot u+2]Q, \pi_p(Q)}(P) \cdot l_{[6\cdot u+2]Q + \pi_p(Q), \pi_p^2(Q)}(P))^{\frac{p^{12} - 1}{r}}.
\]

Where:

- \( r \) is the number of \( \mathbb{F}_p \) rational points in \( E \). That is to say, the number of points in the group or the pairs of points in \( \mathbb{F}_p \) which satisfy the curve equation (This is defined as \( n \) above when discussing BN curves)

- \( f_{s,Q} \) for \( s \in \mathbb{N}, Q \in \mathcal{G}_2 \) is a particular family of normalised \( \mathbb{F}_{p^{12}} \) rational functions. This is referred to as the Miller function (See the text by Silverman [156] for more details)

- \( l_{Q_1,Q_2} \) is the equation of the line corresponding to the addition of \( Q_1, Q_2 \in \mathcal{G}_2 \)
• \( \pi_p : E \to E \) is the Frobenius endomorphism \( \pi_p(x, y) = (x^p, y^p) \). An endomorphism is a mapping from a mathematical object (in this case \( E \)) to itself and the Frobenius endomorphism is simply the mapping which maps every element to its \( p \)th power, where \( p \) is the characteristic of the group (characteristic of a ring is defined as the smallest number of times the multiplicative element must be summed to give the additive identity).

The pairing is calculated using a piece of code executed a certain amount of times in a loop (the Miller loop). The loop order is defined as the number of times the loop must execute to calculate the pairing.

### 2.2. Assumptions

The majority of cryptographic schemes’ security rely on some form of assumption (information theoretically secure scheme being the exception [12,130,153]). The security is shown by reducing the security of the scheme to the assumption. If the security of the scheme is broken then it is also possible to break the underlying assumption. One question which arises is, if the security is just reduced to an assumption, why show its security at all and not just assume that the scheme is secure? By reducing the security to a few assumptions, there are a few assumptions which can be studied in great detail to convince ourselves that it is a hard problem to solve. This is compared to new schemes being deployed which are assumed to be secure without any empirical evidence to back it up.

There are several assumptions that are used by the majority of the literature [3, 154]. In this section the assumptions that will be used for the remainder of the thesis will be presented here. All of the assumptions used within this thesis are assumptions over groups. When a variation of these assumptions is required it will be introduced in the corresponding section and then will be related back to one of these fundamental assumptions, either directly or indirectly.

We start by giving the discrete logarithm assumption, which is the strongest of all of the assumptions and the basis on top of which other assumptions were formalised when they were required.
**Definition 6** (Discrete Logarithm Problem (DL)). Let $G$ be a group, of order $p$, with generator $g$. The discrete logarithm problem is then defined as; given $g, g^x$ find $x$, where $x$ is chosen uniformly at random from $\mathbb{Z}_p$. Given an adversary $A$, the advantage is defined as:

$$\text{Adv}_{G}^{\text{dl}}(A) = \Pr[A(g, g^x) = x].$$

The discrete logarithm assumption is then that the discrete logarithm problem is hard to solve.

In the standard model (see Sect. 2.4 for an explanation) the discrete logarithm is still an assumption; it has not been shown to be computationally difficult. In the GGM it was proven to be hard by Shoup [154].

For lots of cryptographic purposes the discrete logarithm assumption is too strong. Diffie and Hellman constructed a key exchange protocol [51] which is secure under the discrete logarithm assumption. However, the security does not require the full power of the discrete logarithm assumption but a weaker notion which informally states that given two elements it is hard for an adversary to construct a (particular) third element. In the key exchange case the first two elements correspond to the messages passed in the protocol while the third corresponds to the shared key. This was formalised into the Computational Diffie-Hellman Problem.

**Definition 7** (Computational Diffie-Hellman Problem (CDH)). Let $G$ be a group, of order $p$, with generator $g$. The computational Diffie-Hellman problem is then defined as; given $g, g^x, g^y$ find $g^{x\cdot y}$, where $x, y$ are chosen uniformly at random from $\mathbb{Z}_p$. Given an adversary $A$, the advantage is defined as:

$$\text{Adv}_{G}^{\text{cdh}}(A) = \Pr[A(g, g^x, g^y) = g^{x\cdot y}].$$

If an adversary can solve the discrete logarithm problem they can be used to solve the CDH problem [96]. Given $g^x, g^y$ they can calculate $x, y$ since they can solve the discrete logarithm problem and using this they can then calculate $g^{x\cdot y}$.

A decisional variation of the CDH problem was given in which instead of being able to calculate the third value of a Diffie-Hellman triple $(g^x, g^y, g^{x\cdot y})$ the adversary must be able to distinguish the true third element from a random group element.
Definition 8 (Decisional Diffie-Hellman Problem (DDH)). Let $G$ be a group, of order $p$, with generator $g$. The decisional Diffie-Hellman problem is then defined as; given $g, g^x, g^y, g^z$ determine if $g^z = g^{x-y}$ or $g^z = g^r$, where $x, y, r$ are chosen uniformly at random from $\mathbb{Z}_p$ and $z = x \cdot y$ if $b = 1$ and $r$ otherwise for $b$ chosen uniformly at random from $\{0,1\}$. Given an adversary $A$, the advantage is defined as:

$$\text{Adv}^{\text{ddh}}_{G}(A) = \text{Pr}[A(g, g^x, g^y, g^x \cdot y) = 1] - \text{Pr}[A(g, g^x, g^y, g^r) = 1].$$

If an adversary can solve the CDH problem they can be used to solve the DDH problem [96].

Given $g^x, g^y$ they can calculate $g^{x-y}$ and then use this value to compare against $g^z$.

While the decisional Diffie-Hellman problem is assumed to be hard in certain groups, when a pairing is involved, it can become trivial. Consider a (type 1) pairing from $G_1$ to $G_3$, then the decisional Diffie-Hellman problem in $G_1$ is easy. Given $g_1^x, g_1^y, g_1^z$ it can simply be checked whether $e(g_1^x, g_1^y) = e(g_1^z, g_1^x \cdot y)$. This checks if $e(g_1, g_1)^{x-y} = e(g_1, g_1)^z$. Therefore it becomes desirable to introduce some additional hardness assumptions for when working over groups with a pairing.

Definition 9 (Co-Bilinear Diffie–Hellman Problem (CBDH)). Let $B G = (G_1, G_2, G_3, e, p)$ be a set of groups with a pairing between them. The co-bilinear Diffie-Hellman problem is then defined as; given $g_1, g_2, g_2^x, g_2^y$ find $g_3^{x-y}$, where $x, y$ are chosen uniformly at random from $\mathbb{Z}_p$. Given an adversary $A$, the advantage is defined:

$$\text{Adv}^{\text{cbdh}}_{BG}(A) = \text{Pr}[A(g_1, g_2, g_2^x, g_2^y) = g_3^{x-y}].$$

It is apparent that if the adversary can solve the CDH problem they can solve the CBDH problem by computing $g_2^{x-y}$ and calculating the answer by outputting $e(g_1, g_2^{x-y})$.

Definition 10 (Decisional Co-Bilinear Diffie–Hellman Problem (DCBDH)). Let $B G = (G_1, G_2, G_3, e, p)$ be a set of groups with a pairing between them. The decisional co-bilinear Diffie-Hellman problem is then defined as; given $g_1, g_2, g_2^x, g_2^y, g_3^z$ determine if $g_3^z = g_3^{x-y}$ or $g_3^z = g_3^r$, where $x, y, r$ are chosen uniformly at random from $\mathbb{Z}_p$ and $z = x \cdot y$ if $b = 1$ and $r$ otherwise.
otherwise for \( b \) chosen uniformly at random from \( \{0, 1\} \). Given an adversary \( A \), the advantage is defined as:

\[
\text{Adv}^{\text{dcbdh}}_{\text{BG}}(A) = \Pr[A(g_1, g_2, g_2^x, g_2^y, g_2^{xy}) = 1] - \Pr[A(g_1, g_2, g_2^x, g_2^y, g_3) = 1].
\]

An adversary who can either solve the CBDH problem or the DDH problem in \( \mathbb{G}_2 \) will be able to solve the DCBDH problem using techniques similar to those presented above.

The difficulty of the two problems given when a bilinear map is involved, depends on the type of the pairing used. For example, if a type one pairing is involved then \( \mathbb{G}_1 = \mathbb{G}_2 \) and a similar issue arises where the result can be calculated by pairing the results together. A similar issue arises for type two pairings when a homomorphism is known. Thus it is important to take such considerations into account when a scheme is being implemented as to what groups and pairing are chosen.

### 2.3. Security Notions

This section introduces the standard security games, without leakage, that will be used throughout and built upon to define security notions in which the adversary is also allowed to receive some leakage. Where there are well known relationships between two security notions, the relation is also given and in some cases two security notions are shown to be equivalent. The advantage of giving two security notions when they are equivalent tends to be that one is a more intuitive notion to work with while the other may make a reduction cleaner or simpler to write down. Here we focus on the security definitions for PRFs (and PRGs), symmetric encryption and MACs, since these are the three major constructs that will be focused on from this point forward.

Before discussing any schemes or security notions, an overview will be given of how the security experiments work and how they should be understood. All experiments have a function called \( \text{Exp} \) of the form \( \text{Exp}_{\text{a}}(c) \) where \( a \) is the name of the security notion (e.g. IND-CPA), \( b \) is the scheme, primitive etc that is being tested for security property \( a \) (so \( b \) could be an encryption scheme being tested to see if it is IND-CPA secure for example) and \( c \) is any inputs...
to the experiment. The adversary $A$ playing the experiment is always passed in as one of these parameters. If the experiment is an indistinguishability type game (where the adversary must tell which of two worlds they are currently in) a bit denoting the world is also passed in. The experiment can then call any functions (from $b$) and generate any randomness that it requires. However, a requirement is that the experiment must call the adversary $A$ during its execution. This call is of the form $x \leftarrow A^y(z)$. The value $x$ is the output from $A$, while $z$ is the input. The adversary may also have access to some oracles $y(\cdot)$ which they can make queries to. If this is the case the oracles will be written out alongside the experiment. When there are external oracles they can access anything generated by $\text{Exp}$, for example, the oracles can use the information about which of the two worlds they are in, as well as any functions of the scheme $b$. In some situations (such as IND-CCA1) the adversary is called in multiple parts. After the adversary has been called more computation may be performed (such as in the EUF-CMA game the tag output by the adversary is verified) before completion.

The advantage $\text{Adv}_b^a(A)$ is the probability that adversary $A$ wins security game $a$ against scheme $b$.

2.3.1. Pseudo Random Functions and Generators

**Pseudo Random Function (PRF)**

In cryptography a truly random function is desirable. However, it is not possible to instantiate such a construction, as the only way to do this is to store a look up function with a one to one mapping between the input and the output - this would be infeasibly large. Hence the goal of a PRF is to be a, keyed, construction which is efficiently describable and computable which behaves ‘as much like a truly random function as possible’. The security definition below captures what it means for a function to be ‘as much like a truly random function as possible’.

A Pseudo Random Function family, is a family of functions $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$. The key $k$ is chosen uniformly at random from the key space $\mathcal{K}$. The function $F$, defined by the key $k$, is a deterministic function which on input $X \in \mathcal{X}$ outputs an element $Y \in \mathcal{Y}$, written $Y \leftarrow F_k(X)$.

**Definition 11** (Pseudo Random Function (PRF)). Let $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ be a family of functions.
Then Fig. 2.4 defines the PRF security game. The advantage of an adversary $A$ winning the game is defined as $\text{Adv}^{\text{prf}}_F(A) = \Pr[\text{Exp}^{\text{prf}}_F(A, 1) = 1] - \Pr[\text{Exp}^{\text{prf}}_F(A, 0) = 1]$.

In the PRF experiment the adversary has access to a function called RR which will either apply the PRF $F$ or a truly random function to the input, depending if the bit is set to zero or one. The bit $b$ corresponds to if the adversary is in the real or random world. The truly random function has the requirement that if the same input is given twice it returns the same answer (to stop trivial wins). The game does this by randomly generating a response if it is the first time an input has been seen and then storing it in a look up table, so that it can respond with the same answer if the query is repeated.

**Definition 12** (weak Pseudo Random Function (wPRF)). Let $F: \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$ be a family of
functions. Then Fig. 2.5 defines the wPRF security game. The advantage of an adversary $A$ winning the game is defined as $\text{Adv}_{wprf}^F(A) = \Pr[\text{Exp}_{wprf}^F(A, 1) = 1] - \Pr[\text{Exp}_{wprf}^F(A, 0) = 1]$.

It can be shown that a PRF $F$ is also a wPRF. The intuition is that if an adversary cannot win the PRF game where they are allowed to adaptively choose the inputs to the challenge oracle then they will not be able to win the game when they do not have choice of the inputs. However, the converse is not true. For example, it is possible a function that always maps 0 to 0. This function can be a wPRF because the chances of 0 being selected are small but it is clearly not a PRF.

**Pseudo Random Generator (PRG)**

The goal of a PRG is to provide a stream of ‘random looking’ bits such, that given the first $n$ bits of the output, the adversary cannot predict the next output bit (the $(n + 1)\text{th}$ bit). This is captured by the experiment in which, after the adversary has been given the first $n$ bits, they will either receive the next bits as generated by the PRG, or will receive truly random bits. The adversary then has to distinguish between which of the two worlds they are in. If an adversary can predict the next bit then they will be able to win the distinguishing game with a reasonably high probability.

A PRG is a function $G : \mathcal{X} \rightarrow \mathcal{Y}$ such that the size of $\mathcal{Y}$ is considerably larger than the size of $\mathcal{X}$. The function $G$ is a deterministic function which on input $X \in \mathcal{X}$ outputs an element $Y \in \mathcal{Y}$, written $Y \leftarrow G(X)$. To use a PRG to get a stream of output the output distribution can be viewed to be of the form $\mathcal{X} \times \mathcal{X}'$ enabling the state to be updated, with the remaining information output. This new state can be used to recall the scheme and by repeatedly performing this a stream can be output.

**Definition 13** (Pseudo Random Generator (PRG)). Let $G : \mathcal{X} \rightarrow \mathcal{Y}$ be a function. Then Fig. 2.6 defines the PRG security game. The advantage of an adversary $A$ winning the game is defined as $\text{Adv}_{\text{prg}}^G(A) = \Pr[\text{Exp}_{\text{prg}}^G(A, 1) = 1] - \Pr[\text{Exp}_{\text{prg}}^G(A, 0) = 1]$.

In the PRG experiment the adversary has access to two functions RR and Fn. The function
experiment $\text{Exp}^\text{PRG}_G(A, b)$:

1. $\text{flag} \leftarrow \text{false}$
2. $s_0 \leftarrow \mathcal{X}$
3. $b' \leftarrow A^{\text{Fn}().\text{RR}()}$
4. Return $b'$

proc $\text{RR}()$:

1. if $\text{flag} = \text{true}$ then
   1.1. Return $\mathcal{Z}$
2. else if $b = 1$ then
   2.1. $x_{i+1}\parallel s_{i+1} \leftarrow G(s_i)$
3. else
   3.1. $x_{i+1}\parallel s_{i+1} \leftarrow \mathcal{Y}$
4. end if
5. $\text{flag} \leftarrow \text{true}$
6. Return $x_{i+1}$

proc $\text{Fn}()$:

1. if $\text{flag} = \text{true}$ then
   1.1. Return $\mathcal{Z}$
2. end if
3. $x_{i+1}\parallel s_{i+1} \leftarrow G(s_i)$
4. Return $x_{i+1}$

RR behaves similarly to the function in the PRF games except that it may only be called once. This is captured by setting a flag to true once the function has been called and once the flag has been set both RR and Fn no longer return any output to the adversary. The function Fn calculates the next state and output of the PRG from the current state and returns the output to the adversary.

### 2.3.2. Encryption

Informally, the goal of an encryption scheme is to encrypt a message such that it can only be read by someone who has the key, while anyone without the key can learn nothing about the message. However, the question arises that if an adversary is given a ciphertext, what is it acceptable for them to learn about the message and what is unacceptable? Clearly it is unacceptable for the adversary to learn the message (this key goal will be captured by one-wayness), while it is acceptable for them to learn the size of the message. An encryption of the entire Internet and the encryption of a single word are going to have very different sizes, and if they did not then encryption would be extremely inefficient, since all ciphertexts would have to be increased to the size of the largest message that will ever be encrypted. But is it acceptable for the adversary to learn the first bit of the message, or if two ciphertexts contain the same message or should the adversary be able to create a valid ciphertext even if they do not know the underlying message? All of these questions are captured within the following formal definitions for a secure encryption scheme.
There are two types of encryption schemes; symmetric key and asymmetric key. In a symmetric key encryption scheme two parties who wish to communicate share a secret key which is used for both encryption and decryption. We will be focusing on symmetric encryption for the remainder of this work. Asymmetric key encryption is also known as public key encryption. Here there are two keys per person; their public key, which is given to everyone (does not need to be kept secret), which is used to encrypt messages, and the secret key, which is only known by a single user and is used for decryption. The issue with symmetric key encryption is that for every person to be communicated with, a different shared secret key is needed. This means that for $n$ people who all wish to communicate $n \cdot (n - 1)/2$ keys are required. Asymmetric key encryption reduces this to one key pair per person giving $n$ keys. However, due to efficiency reasons, symmetric key encryption tends to be preferred where possible. In this work we focus on symmetric key encryption and thus these are the notions given here.

A symmetric encryption scheme $\mathcal{SE} = (KG, E, D)$ consists of three algorithms; A key generation algorithm which is a randomised algorithm that outputs a key $k$, written $k \xleftarrow{\$} KG$. An encryption algorithm which can be either randomised or stateful which takes in a key $k$ and plaintext $M \in \mathcal{M}$ (from some message space $\mathcal{M}$) and outputs a ciphertext $C$ and is written as $C \xleftarrow{\$} E_k(M)$ or $C \leftarrow E_k(M)$ respectively. Finally, decryption is a deterministic (and stateless) algorithm which takes in a key $k$ and ciphertext $C$ and outputs $M \in \mathcal{M} \cup \{\bot\}$. For correctness, it is required that for all $k \xleftarrow{\$} KG$ and all $M \in \mathcal{M}$ that $D_k(E_k(M)) = M$.

There are three types of ways to generate randomness for an encryption scheme: probabilistic, initialisation vector (IV) and nonce (number only once) based. In probabilistic encryption, the randomness is generated uniformly at random by the scheme. In IV based encryption, the randomness is generated out of band and in the nonce setting the adversary controls the randomness, providing that each value is used only once. The majority of security definitions given here for encryption are for probabilistic encryption. The exception is the authenticated encryption game which is nonce based. Nonce based encryption was formalised comparatively recently and thus the original games were probabilistic. Secondly, the generic compositions we use convert an IV based encryption scheme (which can be seen as a variant of probabilistic encryption in which the randomness is provided by a single IV) into a nonce based authenticated encryption
scheme [122]. However, if nonce based variants of the other games are required, it is fairly trivial to convert the games to support nonce based encryption. Nonce based encryption provides a stronger definition since the adversary can choose the randomness (providing they do not repeat choices) while in probabilistic encryption, it is generated uniformly at random. Formally, a nonce based scheme can be converted to a probabilistic scheme by generating the nonce uniformly at random and passing it into the original scheme. The chance of a nonce repeating is bounded by the birthday bound which is suitably small for a large enough nonce space.

**Indistinguishability**

The indistinguishability notions try to capture the property that an adversary cannot distinguish the encryption of one chosen message from the encryption of another chosen message. This notion also captures all weaker properties such as cannot learn the first bit of the message or how many bits of the message are set to one. There are three variations of the indistinguishability game (and there will be similar variation for other types of games described). Chosen Plaintext Attack (CPA), the adversary only has access to a challenge oracle, while the second two notions are variations of Chosen Ciphertext Attack (CCA1 and CCA2) which allow decryption queries as well. CCA1 only allows decryption before a challenge is received while CCA2 also allows decryption queries after the challenge queries (providing the adversary does not decrypt the challenge).

**Definition 14** (Indistinguishability Under Chosen Plaintext Attacks (IND-CPA)). Let $\mathcal{SE} = (KG, E, D)$ be a symmetric encryption scheme. Then Fig. 2.7 defines the IND-CPA security game. The advantage of an adversary $A$ winning the game is defined as $\text{Adv}_{\mathcal{SE}}^{\text{indcpa}}(A) = \text{Pr}[\text{Exp}_{\mathcal{SE}}^{\text{indcpa}}(A, 1) = 1] - \text{Pr}[\text{Exp}_{\mathcal{SE}}^{\text{indcpa}}(A, 0) = 1]$.
In the IND-CPA experiment the adversary has access to the Left-or-Right oracle \( LR \), which is given two messages and will return either an encryption the left message \( M_0 \) or the right message \( M_1 \). Since the oracle will always return an encryption of the ‘same side’, we say that the bit \( b \) corresponds to if the adversary is in the left world or the right world.

**Definition 15** (Indistinguishability Under Chosen Ciphertext Attacks 1 (IND-CCA1)). Let \( SE = (KG, E, D) \) be a symmetric encryption scheme. Then Fig. 2.8 defines the IND-CCA1 security game. The advantage of an adversary \( A = (A_0, A_1) \) winning the game is defined as

\[
Adv_{SE}^{\text{indcca1}}(A) = \Pr[Exp_{SE}^{\text{indcca1}}(A, 1) = 1] - \Pr[Exp_{SE}^{\text{indcca1}}(A, 0) = 1].
\]

An encryption scheme which is secure in the IND-CCA1 sense will also be secure in the IND-CPA sense [20] (proofs shown for public key cryptography but trivial to convert to symmetric key). The intuition is that if the scheme is secure when the adversary has access to a decryption oracle before they receive a challenge then it will also be secure when this oracle is removed. However, it can be shown that there are schemes which are IND-CPA secure but are not IND-CCA1 secure [20].

**Definition 16** (Indistinguishability Under Chosen Ciphertext Attacks 2 (IND-CCA2)). Let
$\mathcal{SE} = (\mathcal{K}_G, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Then Fig. 2.9 defines the IND-CCA2 security game. The advantage of an adversary $A$ winning the game is defined as $\text{Adv}^{\text{indcca2}}_{\mathcal{SE}}(A) = \Pr[\text{Exp}^{\text{indcca2}}_{\mathcal{SE}}(A, 1) = 1] - \Pr[\text{Exp}^{\text{indcca2}}_{\mathcal{SE}}(A, 0) = 1]$.

The set $S$ contains all ciphertexts that are output from the Left-or-Right challenge oracle and is used to stop challenges being decrypted. If a challenge is sent to decryption, it will be in $S$ and therefore the oracle will return fail $\diamondsuit$ instead of the correct value. This is used to stop trivial wins. Several of the games given below will use sets to avoid certain queries, to avoid trivial wins.

From this point forward IND-CCA is used to denote IND-CCA2.

An encryption scheme that is IND-CCA2 secure will also be IND-CCA1 secure [20]. This is because if an adversary cannot win when they can make decryption queries both before and after the challenge query they will not be able to win when they can only make decryption queries before the challenge query. It is important to note that these notions are not equivalent.

It may seem strange/counter-intuitive that of these three indistinguishability games the only game in which the adversary has direct access to an encryption function is IND-CCA1. This is not because it is the only game of the three where the adversary can ask for encryptions of messages but because it is the only game where it is required to be explicitly given. In the IND-CPA and IND-CCA2 games the encryption oracle can be given explicitly but it is not necessary because if the adversary requires an encryption of a message $M$ they can make a challenge query where both messages are $M$ guaranteeing that they receive and encryption of $M$. However, for the IND-CCA1 game, in the first phase the adversary can make encryption and decryption queries but as soon as a challenge query is made the adversary can no longer make decryption queries. Hence, in this scenario a dedicated encryption oracle is required.

**Real or Random Security**

The real or random security notions try to capture that an adversary cannot distinguish the encryption of a chosen message from the encryption of a random (unknown) message. This notion may initially seem weaker than the indistinguishability notion given previously but it has been shown that the two notions are equivalent.
**Definition 17** (Real or Random Security Under Chosen Plaintext Attacks (RoR-CPA)). Let $SE = (KG, E, D)$ be a symmetric encryption scheme. Then Fig. 2.10 defines the RoR-CPA security game. The advantage of an adversary $A$ winning the game is defined as $Adv_{rorcpa SE}^A = Pr[Exp_{rorcpa SE}^A(A, 1) = 1] - Pr[Exp_{rorcpa SE}^A(A, 0) = 1]$.

In the RoR-CPA experiment the adversary has access to a Real-or-Random oracle $RR$ which has similarities to the oracle for the PRF game. However, instead of returning either the encryption of the message or a random ciphertext (similar to how the oracle behaved in the PRF game), it will return either an encryption of the given message or an encryption of a random message. The bit $b$ denotes if the adversary is in the real or random world.

**Definition 18** (Real or Random Security Under Chosen Ciphertext Attacks (RoR-CCA)). Let $SE = (KG, E, D)$ be a symmetric encryption scheme. Then Fig. 2.11 defines the RoR-CCA security game. The advantage of an adversary $A$ winning the game is defined as $Adv_{rorcca SE}^A = Pr[Exp_{rorcca SE}^A(A, 1) = 1] - Pr[Exp_{rorcca SE}^A(A, 0) = 1]$.
It can be shown that real or random security is equivalent to the indistinguishability notions given above, both for chosen plaintext and chosen ciphertext attacks [19]. The intuition (in one direction) is that if an adversary can distinguish between the encryption of a chosen message and a random message, then they will be able to distinguish between, the encryption of, two chosen messages. The other direction of the equivalence is slightly more complex but follows along similar lines.

**Indistinguishability from Random Bits**

The indistinguishability from random bits notion tries to capture that an adversary cannot distinguish the encryption of a chosen message from a truly random string. This intuitively states that a ciphertext produced by the encryption function maintains no structure.

![image](image.png)

**Figure 2.12.: IND$^*$-CPA experiment**

**Definition 19** (Indistinguishability from Random Bits Under Chosen Plaintext Attacks (IND$^*$-CPA)). Let $\mathcal{SE} = (\mathcal{KG}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Then Fig. 2.12 defines the IND$^*$-CPA security game. The advantage of an adversary $A$ winning the game is defined as $\text{Adv}_{\mathcal{SE}}^{\text{IND}^*\text{CPA}}(A) = \Pr[\text{Exp}_{\mathcal{SE}}^{\text{IND}^*\text{CPA}}(A, 1) = 1] - \Pr[\text{Exp}_{\mathcal{SE}}^{\text{IND}^*\text{CPA}}(A, 0) = 1].$

The IND$^*$-CPA experiment’s oracle $\text{RR}$ behave more closely to that of the PRF game than that of the RoR-CPA game. Given a plaintext it will either return the encryption of the plaintext or a truly random bit string. The bit corresponds to if the adversary is playing in the real or random world. It is possible to tweak this definition slightly such that instead of the random world drawing a random bit string, it samples an element at random from a different set. In certain scenarios this may be a more desirable property, e.g. if ciphertexts were required to resemble credit card numbers or group elements.
**Definition 20** (Indistinguishability from Random Bits Under Chosen Ciphertext Attacks (IND$\$-CCA)).

Let $\mathcal{SE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Then Fig. 2.13 defines the IND$\$-CCA security game. The advantage of an adversary $A$ winning the game is defined as $\operatorname{Adv}^{\text{IND$\$-CCA}}_{\mathcal{SE}}(A) = \Pr[\operatorname{Exp}^{\text{IND$\$-CCA}}(A, 1) = 1] - \Pr[\operatorname{Exp}^{\text{IND$\$-CCA}}(A, 0) = 1]$.

Any encryption scheme that is indistinguishable from random bits is also secure in the ‘original’ indistinguishability notion given above, for both chosen plaintext and chosen ciphertext attacks [140]. The intuition is that if an adversary cannot distinguish ciphertexts from random they will not be able to tell what message is encrypted because encryptions of $M_0$ and $M_1$ will be indistinguishable from each other based on the fact that they are both indistinguishable from random. However, there exists schemes which are indistinguishable but are not indistinguishable from random. Consider an encryption scheme where the ciphertext always begins with a 0. While it may be the case that it is hard to distinguish the encryption of two different messages, it will be easy to distinguish it from a random bit string because, while the ciphertext always begins with 0, a random string will begin with 1 with probability a half.

**Authenticated Encryption**

The previous definitions rely upon probabilistic encryption. However, recently, especially for authenticated encryption, it has been argued that probabilistic encryption is not the most desirable notion. The issue is that probabilistic encryption requires that the randomness is generated uniformly at random and in practice this can be hard to achieve. Nonce based schemes allow the randomness to be chosen by the adversary and passed into the scheme. The only requirement is that a nonce is only used once (for encryption. It can be used many times for decryption).
Therefore nonce based encryption captures truly random values, counters and biased sources of randomness. A nonce respecting adversary is an adversary that only uses each nonce, at most, once for encryption.

The nonce based Authenticated Encryption notion tries to capture both that the adversary cannot distinguish a ciphertext from a random bit string and that they cannot create a new valid ciphertext which was not output from one of the oracles, even when they can control the randomness used by encryption.

\[
\text{experiment } \text{Exp}^{\text{nAE}}(A, b): \\
S \leftarrow \{\} \\
k \leftarrow KG \\
b' \leftarrow A^{\text{RR}(\cdot, \cdot), \text{RP}(\cdot, \cdot)}() \\
\text{Return } b'
\]

\[
\text{proc } \text{RR}(N, M): \\
\text{if } b = 1 \text{ then} \\
C \leftarrow E^N_k(M) \\
\text{else} \\
C \leftarrow \{0, 1\}[E^N_k(M)] \\
\text{end if} \\
S \leftarrow S \cup \{C\} \\
\text{Return } C
\]

\[
\text{proc } \text{RP}(N, C): \\
\text{if } C \in S \text{ then} \\
\text{Return } \perp \\
\text{end if} \\
\text{if } b = 1 \text{ then} \\
M \leftarrow D^N_k(C) \\
\text{Return } M \\
\text{else} \\
\text{Return } \perp \\
\text{end if}
\]

Figure 2.14.: nAE experiment

**Definition 21** (Nonce Based Authenticated Encryption (nAE)). Let \( SE = (KG, E, D) \) be a symmetric encryption scheme. Then Fig. 2.14 defines the nAE security game. The advantage of a nonce respecting adversary \( A \) winning the game is defined as \( \text{Adv}^{\text{nAE}}_{SE}(A) = \Pr[\text{Exp}^{\text{nAE}}_{SE}(A, 1) = 1] - \Pr[\text{Exp}^{\text{nAE}}_{SE}(A, 0) = 1] \).

In the nAE experiment the adversary has access to two oracles RR, RP. The RR oracle behaves exactly as the one in the IND$-\text{CPA}$ game and either returns encryptions of the message or truly random bit strings. The RP oracle either returns the decryption of the ciphertext or always returns the decryption failed symbol \( \perp \). When RR returns valid encryptions, RP returns valid decryptions and when RR returns random bit strings RP states that decryption failed. These are referred to as the real and ideal worlds respectively.

Any encryption scheme, which is a, nonce based, authenticated encryption scheme will also be IND-CCA2 secure, where the IND-CCA2 game has been suitably adjusted to support nonce based encryption schemes instead of probabilistic schemes [141]. The intuition is that nAE
provides IND-CPA and INT-CTXT security (defined below) which gives IND-CCA2 security. The arguments as to why nAE provides these two security properties is similar to previous arguments and thus not given. However, a nonce based scheme which is IND-CCA2 secure is not guaranteed to be nAE secure (consider a similar example used to the one which argues why IND-CPA is not IND$^*_S$-CPA).

**One-Wayness**

One-wayness is perhaps the weakest of the notions listed here but is also necessary for any other security to be possible. One-wayness states that given a ciphertext of a random message, an adversary cannot learn the underlying plaintext.

**Definition 22** (One-Wayness Under Chosen Plaintext Attacks (OW-CPA)). Let $\mathcal{S}\mathcal{E} = (\mathcal{K}\mathcal{G}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Then Fig. 2.15 defines the OW-CPA security game. The advantage of an adversary $A$ winning the game is defined as $\text{Adv}^{\text{owcpa}}_{\mathcal{S}\mathcal{E}}(A) = \Pr[\text{Exp}^{\text{owcpa}}_{\mathcal{S}\mathcal{E}}(A) = 1]$.

**Definition 23** (One-Wayness Under Chosen Ciphertext Attacks (OW-CCA)). Let $\mathcal{S}\mathcal{E} = (\mathcal{K}\mathcal{G}, \mathcal{E}, \mathcal{D})$ be a symmetric encryption scheme. Then Fig. 2.16 defines the OW-CCA security game.
rity game. The advantage of an adversary $A$ winning the game is defined as $Adv_{SE}^{owcca}(A) = \Pr[\text{Exp}_{SE}^{owcca}(A) = 1]$.

Any scheme which is OW-CCA secure is also OW-CPA secure. The converse is not true.

Any scheme which is secure in the indistinguishability sense is also secure in the one-wayness sense. The intuition is that if a scheme is not one-way secure then an adversary could send two messages in the indistinguishability game, receive a ciphertext from which they can learn the underlying message and thus win the indistinguishability game. Therefore, any scheme which is secure in the indistinguishability sense is also secure in the one-wayness sense. It is possible to have a scheme which is one-way secure but not secure in the indistinguishability sense. Consider an encryption scheme which reveals the first bit of the message. This bit makes it easy to win the indistinguishability game but only gives the adversary a single bit of the message so may not make one-wayness easy to solve.

**Integrity**

The integrity notions try to capture that an adversary cannot create a ciphertext of a message which they have not already seen the encryption of.

```plaintext
experiment Exp_{SE}^{intptxt}(A):
  $k \gets KG$
  $win \gets 0$
  $S \gets \{\}$
  $A^{Enc(\cdot), Dec(\cdot)}()$
  Return $win$

proc Enc($M$):
  $C \gets E_k(M)$
  $S \gets S \cup \{M\}$
  Return $C$

proc Dec($C$):
  $M \gets D_k(C)$
  if $M \neq \perp$ and $M \notin S$
    then $win \gets 1$
        Return 1
  end if
  Return 0
```

Figure 2.17.: INT-PTXT experiment

**Definition 24** (Integrity of Plaintexts (INT-PTXT)). Let $SE = (KG, E, D)$ be a symmetric encryption scheme. Then Fig. 2.17 defines the INT-PTXT security game. The advantage of an adversary $A$ winning the game is defined as $Adv_{SE}^{intptxt}(A) = \Pr[\text{Exp}_{SE}^{intptxt}(A) = 1]$. 

53
experiment $\text{Exp}^{\text{intctxt}}_{\mathcal{S}E}(A)$:

$\begin{align*}
& k \overset{\$}{\leftarrow} KG \\
& \text{win} \leftarrow 0 \\
& S \leftarrow \{\} \\
& A^{\text{Enc}(\cdot), \text{Dec}(\cdot)} \\
& \text{Return } \text{win}
\end{align*}$

proc $\text{Enc}(M)$:

$\begin{align*}
& C \leftarrow E_k(M) \\
& S \leftarrow S \cup \{C\} \\
& \text{Return } C
\end{align*}$

proc $\text{Dec}(C)$:

$\begin{align*}
& M \leftarrow D_k(C) \\
& \text{if } M \neq \perp \text{ and } C \notin S \\
& \quad \text{then} \\
& \quad \text{win} \leftarrow 1 \\
& \quad \text{Return } 1 \\
& \text{end if} \\
& \text{Return } 0
\end{align*}$

Figure 2.18.: INT-CTX experiment

**Definition 25** (Integrity of Ciphertexts (INT-CTX)). Let $\mathcal{S}E = (KG, E, D)$ be a symmetric encryption scheme. Then Fig. 2.18 defines the INT-CTX security game. The advantage of an adversary $A$ winning the game is defined as $\text{Adv}_{\mathcal{S}E}^{\text{intctxt}}(A) = \Pr[\text{Exp}_{\mathcal{S}E}^{\text{intctxt}}(A) = 1]$.

Any encryption scheme which provides integrity of ciphertexts also provides integrity of plaintexts, while the converse is not true [21]. This can be seen because, if an adversary cannot win the INT-CTX game where they can provide a ciphertext of a message they have queried before then they will not be able to win the INT-PTXT game where the message must be a new one.

It can be shown that an encryption scheme which is both IND-CPA secure and INT-CTX secure is also IND-CCA secure [21]. The intuition is that for an adversary to win the IND-CCA game they must either distinguish the challenge without the help of decryption (thus winning the IND-CPA game) or be able to send a new ciphertext to the decryption oracle (breaking INT-CTX). Giving the desired result. This is not an equivalence because it can be shown that IND-CCA does not provide INT-CTX. Consider the encryption scheme in which the all zero string is always a valid ciphertext (for example of the all zero plaintext). The INT-CTX game is then trivial to win because the adversary can ask for a decryption of the all zero string. However, for the IND-CCA game this will help the adversary only if they send the zero message and get back an encryption which is the all zero string which happens with suitably small probability.

### 2.3.3. Message Authentication Codes (MACs)

An encryption scheme provides no guarantees as to where the message came from and that it has not been tampered with (only that information about the message can’t be learnt). Consider
the one time pad which provides (perfect) encryption. It is still possible to change the under-lying message by flipping bits of the ciphertext (by the exclusive-or nature of a one time pad, which flips the corresponding bits of the plaintext). There are many situations in practice where guarantees are required that a received message was written by the individual who claimed to have sent it. For example, consider a message from Alice to her bank stating “Pay Bob £10”, the bank want to know that this message in fact came from Alice and that it cannot be changed to read “Pay Bob £10,000”. In the scenario where two parties share a secret key and want to check that the message received was sent by the other party and has not been tampered with en route, a MAC is required. The formal definitions for the security requirements of a MAC are given below.

A MAC $M = (KG, Tag, VRFY)$ consists of three algorithms; A key generation algorithm which is a randomised algorithm that outputs a key $k$, written $k \xleftarrow{\$} KG$. A tagging algorithm which can be either randomised or stateful which takes in a key $k$ and plaintext $M \in \mathcal{M}$ (from some message space $\mathcal{M}$) and outputs a tag $\sigma$ and is written as $\sigma \xleftarrow{\$} Tag(k, M)$ or $\sigma \leftarrow Tag(k, M)$ respectively. Finally verify is a deterministic (and stateless) algorithm which takes in a key $k$, message $M$ and tag $\sigma$ and outputs $b \in \{0, 1\}$ depending if $\sigma$ is a valid tag of $M$. For correctness it is required that for all $k \xleftarrow{\$} KG$ and all $M \in \mathcal{M}$ that $VRFY(k, M, Tag(k, M)) = 1$.

**experiment** $\text{Exp}_{\text{eufcma}}^M(A)$:

$k \xleftarrow{\$} KG$

$S \leftarrow \{}$

$(M^*, \sigma^*) \leftarrow A^{\text{Tag}(\cdot), \text{Verify}(\cdot)}()$

if $M^* \in S$ then

Return 0

end if

Return $\text{VRFY}(k, M^*, \sigma^*)$

**proc** $\text{Tag}(M)$:

$S \leftarrow S \cup \{M\}$

$\sigma \leftarrow Tag(k, M)$

Return $\sigma$

**proc** $\text{Verify}(M, \sigma)$:

$b \leftarrow \text{VRFY}(k, M, \sigma)$

Return $b$

Figure 2.19.: EUF-CMA experiment

**Definition 26** (Existential Unforgeability Under Chosen Message Attack (EUF-CMA)). Let $M = (KG, Tag, VRFY)$ be a MAC. Then Fig. 2.19 defines the EUF-CMA security game. The advantage of an adversary $A$ winning the game is defined as $\text{Adv}_{\text{eufcma}}^M(A) = \text{Pr}[\text{Exp}_{\text{eufcma}}^M(A) = 1]$.  

55
Definition 27 (Strong Existential Unforgeability Under Chosen Message Attack (sEUF-CMA)).

Let $M = (\text{KG}, \text{Tag}, \text{VRFY})$ be a MAC. Then Fig. 2.20 defines the sEUF-CMA security game. The advantage of an adversary $A$ winning the game is defined as $\text{Adv}_{\text{seufcma}}(A) = \text{Pr} [\text{Exp}_{\text{seufcma}}(A) = 1]$.

Given a MAC which is sEUF-CMA secure, it will also be EUF-CMA secure [21] (EUF-CMA was originally denoted weak EUF-CMA). The intuition as to the relation between these security statements is that if an adversary cannot create a forgery when they can forge on a previously used message, they will not be able to create a forgery when it has to be on a previously unseen message. The converse of this statement can be shown to be false.

2.4. Models

There are a variety of models used within the cryptographic literature [22, 48, 117, 123, 154]. While the goal is to be able to prove schemes secure in the standard model, this is not always possible and sometimes alternative models are required. As an example, the discrete logarithm problem does not have a proof that it is hard in the standard model but can be proven secure in the GGM [154]. The various models are used to capture different aspects, be this how an adversary can interact with the underlying primitives or that randomness can be generated. Unfortunately there have been separation results which show that it is possible to construct schemes within some of these models such that however they are instantiated in the standard model they are insecure [41,49]. This implies that care must be taken when instantiating schemes shown secure in these models. However, these models are still a useful tool to give a ‘feel’ for security when
a proof (or insecurity) in the standard model is not known.

2.4.1. Standard Model

In the standard model the only assumptions/restrictions made are on the computational power and time of the adversary. It can be seen that these conditions are necessary otherwise the majority of cryptography would not be possible, perfect security being the notable exception. For example, if a discrete log adversary was not computationally bounded they could test all exponents before outputting the correct answer. Other than this there are no assumptions placed on the adversary or on the set up. For example, there is no assumption on the existence of truly random functions or any shared strings between parties.

The lack of assumptions in this model makes it the most desirable model to prove the security of schemes in. Since all other models also require these assumptions (with the exception of information theoretic adversaries which will not be considered in this work) any proof in the standard model will also hold within the other models. Within this work proofs will be in the standard model unless otherwise stated.

2.4.2. Generic Group Model (GGM)

The goal of the GGM [117, 123, 154] is to restrict the adversary in such a way that the structure of the underlying groups cannot be exploited, beyond what follows from the standard group axioms. The advantage of the GGM is that bounds on how long an algorithm must take can be given, when no such bound is known in the standard model [154]. Once a bound has been given, it is sometimes possible to give a bound in a model which is closer to how a group behaves in practice (such as the Semi-Generic Group Model [93]). The GGM is well established to prove the security of various protocols [38, 148, 157].

There are three descriptions of the GGM, two of which are more common in the literature [117, 123, 154]. In Maurer's description of the model [117] each element can be seen as being stored in an array such that the adversary only has access to ‘handles’ of the array elements. The adversary can provide handles along with an operation to be performed on the elements stored in that location. The model will perform the operations on the element and
store the result in a new location, returning the new handle to the adversary. Since each element is stored in a new location an equality operator is also required so the adversary can check if two items represented by two handles are equal. Shoup [154] represents elements as random bitstrings called encodings (such that each element has a unique bitstring). There is an oracle for each group operation which takes in the encodings of elements, performs the operation and returns the encoding of the result. Since the adversary is given the encodings they are able to tell when two values are equal and thus an equality oracle is not required. We will not discuss the work of Nechaev [123] here since it focuses on proving lowerbounds within the model instead of formalising the model. For the remainder of this work we opt for Shoup’s approach of modelling generic groups. The advantage of Shoup’s model for our work is that an encoding is something that makes sense to leak on, while a handle does not because each element does not have a unique handle.

In the Generic Bilinear Group Model (GBGM) [32] each of the three groups has its own randomised encoding. Each of these encodings will be represented by an injective encoding function \( \xi_1 : \mathbb{Z}_p \to \Xi_1, \xi_2 : \mathbb{Z}_p \to \Xi_2, \xi_3 : \mathbb{Z}_p \to \Xi_3 \) for \( G_1, G_2, G_3 \) respectively, where \( \Xi_1, \Xi_2, \Xi_3 \) are (disjoint) sets of bit strings. The adversary has access to the following 4 oracles:

- \( O_1(\xi_1(a), \xi_1(b)) = \xi_1(a + b \mod p) \)
- \( O_2(\xi_2(a), \xi_2(b)) = \xi_2(a + b \mod p) \)
- \( O_3(\xi_3(a), \xi_3(b)) = \xi_3(a + b \mod p) \)
- \( O_e(\xi_1(a), \xi_2(b)) = \xi_3(a \cdot b \mod p) \)

for all \( a, b \in \mathbb{Z}_p \). Each of the 4 oracles will return \( \bot \) if either of the inputs is not an invalid encoding of an underlying group element. \( O_1, O_2, O_3 \) perform the group operations of \( G_1, G_2, G_3 \) respectively, while \( O_e \) performs the pairing operation. To work with these groups an adversary only needs to be given \( \xi_1(1) \) and \( \xi_2(1) \) (corresponding to the generators of \( G_1 \) and \( G_2 \) respectively) plus access to the four oracles, from which any group element can be computed.

It has been shown that certain schemes shown to be secure in the GGM are insecure in the standard model regardless of how they are instantiated [49].
There are several common techniques for proofs within the GGM. We discuss the ones which will be used within this work here. The first is to represent the group elements, representing variables within the scheme, as indeterminants. We will use the font $\mathcal{X}$ to denote an indeterminant. In this manner, group operations correspond to operations on polynomials. The indeterminants are assigned values as soon as they are required. The next step is to not assign values to the indeterminants until the end of the experiment. This has two advantages; it makes it more apparent that the proof will hold for all variable assignments and secondly (and more importantly) it makes it clear that an adversary’s decisions cannot depend on the value of the group elements. The adversary cannot know what encoding maps to what group element since the polynomials are not evaluated until after the adversary has finished playing the security game. The downside of this method is that if two (distinct) polynomials collide when evaluated the simulation is not consistent because there exists a group element which has two distinct encodings. The Schwartz-Zippel lemma bounds the probability of this happening.

**Lemma 1** (Schwartz-Zippel lemma). Let $P$ be a non-zero, polynomial of degree $d$ over field $\mathbb{F}$ and let $S$ be a finite subset of $\mathbb{F}$. If $x_1, \ldots, x_n$ are chosen independently and uniformly at random from $S$, then
\[
\Pr[P(x_1, \ldots, x_n) = 0] \leq \frac{d}{|S|}
\]
where $|S|$ denotes the size of $S$.

By defining $P = P_1 - P_2$, the Schwartz-Zippel lemma applied to $P$ gives the probability that $P_1(x_1, \ldots, x_n) = P_2(x_1, \ldots, x_n)$

**2.4.3. Random Oracle Model**

In the random oracle model there is a function $H : \{0, 1\}^* \rightarrow \mathcal{D}$ for some domain $\mathcal{D}$ such that on an input it returns a (truly) random result from the output domain. If a query is repeated, the oracle $H$ will return the same answer each time. The random oracle model was first used in a formal manner by Bellare and Rogaway [22].

The random oracle model is used to provide the existence of an efficient truly random function. A common use of random oracles in the literature is as hash functions [45, 151] (a deter-
ministic function that takes in an arbitrary string and returns a fixed sized string) or for the PRF switch used in proofs, when the required properties are hard to prove for known functions. This leads to a proof in the random oracle model instead of the standard model. When a scheme is constructed the random oracle it will have to be instantiated with a concrete function. However, there are schemes that are known to be secure in the random oracle model which are not secure when the random oracle is instantiated with any concrete function [41].

A refinement of the random oracle model is the programmable random oracle model. In this model, providing the output distribution of random oracle is uniform across $D$, then the oracle may choose to map certain input values to certain output values dynamically. This allows reductions to embed challenges within the random oracle. By embedding the challenge in one of the random oracle queries the adversary makes, if the adversary can win their security game then this can be used to ‘solve’ the original challenge embedded into the random oracle. This embedding technique is an important tool used in many security proofs, most of which do not have an alternate proof without this technique. Fischlin et al. [69] show a separation between the random oracle model (without programmability) and the programmable random oracle model. They do this by presenting schemes which can be shown secure in the programmable random oracle model but cannot be proved secure in the random oracle model.
3. Previous Work

While side channel countermeasures were designed to enable the device in the physical world to behave more like the security model, recently theoreticians have come up with new security models which try to capture all, or some, of the side channel attacks, bringing the security models closer to the real world. In this chapter some of the theoretical models are discussed, including what side channels they protect against, the advantages and disadvantages of each model, as well as how the models compare against each other. There have been similar literature reviews of the various leakage models and schemes before. Pietrzak [134] gives a high level description of many of the models which are given below, including private circuits, the bounded retrieval model and auxiliary input. Alwen et al. [6] focus on reviewing the literature for the relative leakage model and the bounded retrieval model. The relative leakage model is fairly uncommon within the literature and thus will not be discussed below. However, the model allows the adversary to acquire an amount of leakage relative to the secret size, regardless of the key size. For example, they can learn 40% of the bits of the secret key. While there have been several literature reviews of schemes that are secure in the face of leakage and the models which security are proved in, there are several advantages to providing a literature review. Firstly, it allows the rest of the work to be put into context. By discussing the advantages and disadvantages of each model here, it helps to justify the choice of leakage models for the remaining chapters. Secondly, all the published literature reviews are over five years old and thus this provides a more current view of the area.
3.1. Simulatable Leakage

The simulatable leakage model works as follows. If it is possible to construct a simulator for the leakage, which does not have access to the secret information, such that a distinguisher cannot differentiate the output of the simulator from the leakage of the device, then clearly the leakage cannot contain any information about the secret key. This model’s design strategy is similar to that of zero knowledge [82].

This model was first looked at, in terms of a PRF, by Standaert et al. [158] where they define a security notion for simulatable leakage before constructing a simulator for AES and using it to construct a leakage resilient PRG. In Ch. 4 this model and scheme is discussed in greater detail together with our follow up paper [108]. In the follow up work it is shown how to adjust the security model such that it better models how an adversary behaves in practice. In this new model an attack is given against the simulator for AES.

Fuller and Hamlin [71] take the simulatable leakage model and expand it into two different models. The first is known as weakly simulatable in which the simulated leakage must be consistent with only public information. The second model is denoted strongly simulatable in which the leakage must be consistent with both the public and secret information. Hence, in this model the distinguisher is also given the secret information.

This model has several advantages over the Standaert et al. [158] model. Firstly, while the original model was specialised to deal with symmetric-key constructions, the new model is more generalised and can deal with any form of construction. However, due to these generalisations, it can no longer deal with the leakage on the key update which was allowed originally. Not only this, but the model in fact only allows leakage on a single key, so can be deemed slightly weaker in this regard. Another restriction in the original simulatable leakage model (while standard in most leakage models) is that leakage is provided with each call to the block cipher. The new model generalises the leakage and allows options such as multiple leakages per blockcipher calls (to represent taking multiple measurements at once) or on the other extreme multiple block cipher calls for only a single leakage function. Finally, the model of Standaert et al. does not allow for the simulator to maintain state, which is limiting, and thus the model of Fuller and Hamlin allows for this.
The advantage of the simulatable leakage model is that creating a simulator for schemes that are used in practice provides these schemes with some form of security guarantees without having to replace currently deployed systems.

3.2. Private Circuits

Ishai et al. [92] introduce the notion of private circuits, where the model allows the adversary to probe up to \( t \) wires (these may vary between invocations) within the circuit representation and learn the values on these wires.\(^1\) The ideal would be for the adversary to probe all wires but the obfuscation impossibility result [13] implies that security in this situation is impossible, and thus \( t \) must be bounded. They then show how to create a provably secure compiler that can take in any circuit \( C \) composed of AND and NOT gates and convert it into a circuit \( C' \) in which the adversary can probe \( t \) wires, yet cannot learn anything more than if they had only access to the function in a black box manner. The scheme works by, additively, secret sharing out all internal wires using \( t + 1 \) shares. Since the adversary can only learn, at most, \( t \) shares, the values on each wire remains information theoretically hidden. By showing how to perform AND and NOT gates on data secret shared in this manner, gives a compiler for the scheme. However, if the circuit \( C \) has \( \mathcal{O}(n) \) gates, then the resulting circuit \( C' \) will have \( \mathcal{O}(t^2 \cdot n) \) gates.

Faust et al. [68] show how to generalise the leakage model, using a leak-free component. The first model looked at is all leakage functions from the complexity class \( \text{AC}^0 \), the class of constant depth circuits. Hence, leakage no longer has to be on (a bounded number of) independent wires but can be a circuit on all wires. The reason that the complexity class cannot be more general than \( \text{AC}^0 \) is that anything more would allow the leakage function to reconstruct the secret from the shares, thus making a break trivial. It is also apparent that (like other models, which will be discussed in the following sections) the output must be bounded in length, otherwise the circuit could return the entire internal state. The second leakage model considered is noisy leakage. In the noisy leakage model the adversary receives all internal wires after the values have had

\(^1\)It is important that the correct model of computation is chosen for each of the leakage models that follows. Micali and Reyzin [120] give a computational model that is suitable for allowing leakage. In this model they define physical virtual-memory computers which are formed form a series of (independent) Turing machines (one per “component” of the computation), each complemented with the ability to leak.
binomial noise added to them. Noise is added by flipping a biased coin (with probability $p$ of a head) and if the result is a head, the value on the wire is negated. This represents measuring a device using a noisy channel such as power. While the leak-free component is small, computation independent and stateless, and gives the ability to generalise the leakage function, a leak-free component is unrealistic because it is hard (or costly) to protect a component such that it does not leak any information.

Rothblum [142] shows how using a new assumption it is possible to construct a scheme secure against leakage from the complexity class $\text{AC}^0$ without requiring a leak free component. The assumption has connections to open problems within complexity theory.

Building on previous work Duc et al. [58] show that if a scheme is secure against probing attacks then it is secure against noisy leakage attacks. The advantage of the noisy leakage model is that it is very close to how an actual attack works when power traces are recorded. The work by Duc et al. implies that the circuit compiler discussed above is also secure against noisy leakage attacks. While Faust et al. [68] show how to provide security against noisy leakage directly, the construction was more complex than that of Ishai et al. [92] and thus this allows the simpler construction to be used.

The majority of circuit compilers either split the state into two shares and require a leak-free component [62, 97] or remove the requirement for the leak-free components by sharing the state into three or more shares [92]. Since it is known to be impossible to have no leak free component and a single share [13], the ideal is to have the secret split into two shares without the requirement of a leak-free component. Dachman-Soled et al. [47] show that certain leak-free components can be replaced with a circuit shared in two parts using deniable encryption and non-committing encryption, in a manner which allows the computation to be leaked upon. There exists schemes in the literature [62, 97] which operate on two shares and require a leak-free component, such that the leak-free component meets the requirements of Dachman-Soled et al. which enables a two component scheme without a leak-free component to be constructed.

The statistically private circuit model [92] can be seen as a middle ground between the private circuit and noisy leakage models. In this model each wire in the circuit is corrupted with independent probability $\rho$. If the wire is corrupted, the adversary receives the value on the wire
and if the wire is not corrupted the adversary receives no information about the wire’s contents. In this model an adversary is expected to see $\rho \cdot n$ wires. Since, the wires are corrupted randomly instead of being chosen adversarially, as in the private circuit model, security is easier to achieve [92].

Exposure Resilient Cryptography [57] can be seen as a restriction to the private circuit model in which the adversary can learn some of the bits that are supposed to be secret; All-or-nothing transforms protect the input given partial knowledge of the output, while exposure-resilient functions do the opposite; protect the output given partial knowledge of the input.

The advantage of the private circuits model is that it allows schemes to be proven secure against noisy leakage. When (power) traces are recorded from a device they are noisy and this noise tends to be Gaussian. Thus this particular model is suitable for capturing the types of measurement adversaries use in practice to attack a device. The probing model itself tends to be less directly useful since it is rare for an adversary to have access to a device in such a manner where they can only probe a fixed amount of wires. One disadvantage of the work, discussed so far, in this model has been the use of the leak free component as this does not capture how a device behaves in practice but recently schemes are being constructed in this model which are more efficient and/or behave in a manner which is closer to practice [17, 42, 63]. Dziembowski et al. [63] remove the non-optimality from the reduction between the noisy leakage model and the probing model by introducing a model denoted the average probing model. This work allows the security of new masking schemes to be shown. Belaïd et al. [17] propose several algorithms for multiplication in the private circuit model and investigate the amount of randomness required in these models. Carlet et al. [42] investigate trading non-linear multiplications for low-degree functions and investigate how to secure such functions within the probing model. This recent flurry of work shows that this model has become the model of choice within the last year for schemes which are both provably secure and practically viable.
3.3. Bounded Storage Model

The bounded storage model was introduced by Maurer [116] for considering provably secure randomized ciphers. The goal was to construct a symmetric key cipher in which there is a large publicly available random string, which is also accessible to the adversary. The assumption made in the model is that the adversary has restricted memory (but not restricted computational power) such that they are not able to access the whole public string. In this model the proposed cipher is shown to be secure.

The work laid out by Maurer [116] was generalised to become the bounded storage model in which, for the desired protocol, all parties (including the adversary) have access to a large random public string, which they can access a small amount at a time. Security is provided against adversaries which have a fair amount of storage (sometimes even more than the honest parties) providing that they cannot store the whole string. The adversary is allowed unlimited computational power.

One particular interesting use of the bounded storage model is for non-interactive timestamping protocols [121]. A (interactive) timestamping protocol is a protocol in which a user can timestamp a document at time $t$ and at any later date it can be verified that the document was timestamped at time $t$, thus proving that the document existed at time $t$. All of these protocols require interacting with a central authority during the time stamping process, while a non-interactive timestamping protocol does not have this requirement. While it has been shown that non-interactive time stamping is, in general, impossible (it should be fairly straight forward to see that without interaction an adversary can timestamp something at an instance in time stating that it was timestamped in the past), Moran et al. [121] show that it can be done in the bounded storage model.

While the bounded storage model was not designed as a model to capture leakage but as a computational model, it can easily be seen how it can correspond to a desirable leakage model. Instead of the large random string being public, it corresponds to some of the secret information held by the honest party and the small amount of the string the adversary accesses corresponds to leakage received via some side channel. It can be seen that the reason the adversary could only access a small portion of the random string is because they only had access to the device.
storing the string for a short amount of time. This is a reasonably standard assumption within the side channel community. However, this model will only allow leakage on the random string and not any other secret information kept by the honest parties.

There has been several other works within this model including key agreement [39] and oblivious transfer [52]. Dziembowski and Maurer [64] show that for key agreement it is not possible to securely agree the initial secret, used to generate the ephemeral key, in the bounded storage model without the two parties, who are agreeing upon the key, having memory size considerably larger than the adversary. Since this is unrealistic, it demonstrates the optimality of the key agreement protocol [39].

### 3.4. Bounded Retrieval Model

A variation of the bounded storage model is the bounded retrieval model [50]. In the bounded retrieval model the adversary is allowed to access a fixed amount of information about the large secret state. This corresponds to situations where the adversary has access to a side channel or has access to the network for a short period of time (intrusion detection software may prevent prolonged connections) and thus can only retrieve a small amount of information about the secret state. However, while the adversary still only has a bounded amount of information he is computationally unbounded. This is different from the bounded storage model where there was the concept of a secret key (e.g. in the symmetric encryption case) that the adversary could not get access to. While in the bounded retrieval model the adversary can request access to any secret information as long as they do not request access to more information than allowed.

Di Crecenzo et al. [50] create password protocols within this model such that an offline attack is no more efficient than an online attack. A password protocol is a protocol which takes a password from a user and allows the user to log in if the password matches. Password protocols tend to be constructed by storing the hash of a password in a database and comparing the hash of the given password to the one stored within the database. Normally if an adversary is allowed access to (even a single row in) the database they can perform an offline attack by trying all possible passwords until the hashes match (this can be achieved since the adversary is computationally
unbounded) and then log in with the matching password. This attack only uses a single password check and thus is considerably more efficient, in terms of queries, than an online attack which keeps trying passwords until the log in is successful. In the bounded retrieval model Di Crecenzo et al. [50] show how to prevent this attack and guarantee that offline attacks cannot be more efficient than an online attack.

The major disadvantage of the bounded retrieval model, and the schemes shown to be secure within it, is the large amount of randomness that is required to produce the secret key. Durnoga et al. [60] propose a solution to this issue in the random oracle model using weakly random data stored on the machine already such as music, emails and photos. Therefore, the key does not have to be stored but can be calculated as required providing the data used on the machine is static. However, this introduces a few of problems. Firstly, the data is only weakly random, while the contents of an email or image may be secret, they will all contain header information of a known standard form. While this standard information is known to an adversary, it should not be the case that this translates into the adversary learning information about the generated secret key. The second issue is that, since the key is being generated using personal files, any leakage on the key should not reveal any information about the underlying personal files. By hashing together multiple blocks both of these problems are circumvented; with enough blocks added to the hash function there is guaranteed to be enough entropy for the output to be suitably random and since the hash is a one way function (modelled as a random oracle), it will not reveal any information about the underlying private files.

This model captures attacks such as password database breaches, where the adversary gets some of, but not all, of a large volume of data and then the adversary can use this information to try and defeat the system. This model does not capture the kind of leakage that is produced by small devices when side channels are measured but more models large data breaches. Since we are interested in side channel leakage within this work, this will not be a suitable choice of model.
3.5. Bounded Leakage Model

The bounded leakage model can be seen as a generalisation of the bounded retrieval model. Instead of being able to retrieve a bounded amount of the secret state, the adversary can pass in a function, have the function evaluated on the whole secret state and then receive the answer, provided that the answer is within the total leakage bound. This more accurately models attacks like DPA, where, instead of parts leaking perfectly and some parts not leaking at all (as in the bounded storage model), the power trace is a, noisy, function of all the secret data.

Unlike the previous two models, the adversary and the leakage functions in this model have to be computationally bounded. This can be seen as a trade off between the two models; the adversary is computationally bounded in return for having more powerful leakage functions. The bounded leakage model more closely models a real life adversary; they have a finite amount of time they can spend trying to attack a device but can vary their measurement set up to capture the information of most use.

The standard usage of the model [137] allows the adversary to provide a leakage function with an arbitrary length output providing that the total leakage received is lower than the total leakage bound. Due to this, the overall leakage bound must be less than the length of the secret key $sk$ otherwise the first query could be $l(sk) = sk$ which would completely break the security of the scheme, yet the leakage would be within the bound.

Compared to the models seen previously, allowing the adversary to adaptively choose arbitrary leakage functions, is closer to how an adversary would interact with a device in practice. However, this is where the similarity between theory and practice ends. In practice, each time a device performs a cryptographic operation, it will leak more information, in the form of power traces, EM traces, etc. However, the bounded leakage model does not capture this continual form of leakage but after the device has leaked a certain amount it stops leaking completely. It can be argued that, since a device’s only secret is the key $sk$ it cannot leak more information than the length of the key and thus this model is an accurate representation of real world leakage. However, this is only true if the goal is key recovery. If the goal is instead something more akin to forging a signature or differentiating between two encrypted message distributions this is no longer the case. Another argument that can be made is that the amount of leakage is bounded
because an adversary only has access to the device for a finite amount of time, in which they can perform side channel attacks. After this the device returns to its owner, at which point the adversary can only see communication go ‘over the wire’ and can no longer measure side channels. This model can be seen to accurately capture this particular scenario.

The question of security when arbitrary, yet bounded, information leaks has been considered for a long time before it was formalised into the bounded leakage model. In 1983 Ben-Or et al. [23] considered the security of RSA when one of the following questions could be asked; given a, non-negligible, interval $I$ and a ciphertext $c$ for some plaintext $x$, is $x \in I$? Given $c$ what is the $i$th bit of $x$? Given $c$ guess the least significant bit of $x$ with error $\frac{1}{4} - \epsilon$. It is shown that, with the oracle which returns a single bit of leakage, it is possible to completely decrypt the message $c$ breaking the security of RSA.

Since the formalisation of this model there has been a host of work within it including; signature schemes, public and symmetric IND-CPA and IND-CCA encryption and Identity Based Encryption [100, 105, 137].

Hazay et al. [89] produce a wide variety of schemes within the bounded leakage model using only minimal assumptions. The schemes created include; public-key encryption secure in the bounded leakage model from a public key encryption scheme, as well as a weak Pseudo Random Function (wPRF), symmetric-key encryption scheme and a MAC all secure within the bounded leakage model from a one-way function. The advantage of this work is that it is the first to create symmetric-key schemes which do not rely on public-key assumptions, as well as the first work to create public-key schemes from search assumptions such as CDH or factoring. Using these, more desirable, weaker assumption comes with the disadvantage that the leakage is less than schemes which use stronger assumptions.

### 3.6. Continuous Leakage Model

The continuous leakage model can be seen as a further generalisation of the bounded leakage model. In this model, each time the adversary makes an oracle call they also pass in a leakage function $l_i$ and receive $l_i(k, x, R)$, for secret information $k$, oracle input $x$ and internal ran-
domness $R$. Unlike the bounded leakage model, the overall total leakage can be unbounded providing the leakage per function call is bounded (otherwise $l_i$ could leak the secret information). However, because the $l_i$’s are adaptive, even if the leakage functions can only leak one bit per call, over time the entire secret will be revealed, since $l_i$ can leak the $i$th bit of the secret $k$. Hence, the key has to be updated in some manner. For example, it is put through a hash function so that $k_{i+1} = H(k_i)$. Therefore, after a key has been used, it cannot be leaked on again. However, it is still possible to entirely leak a key; let $H^n$ denote applying $H$ $n$ times, then if a key $k_i$ has $n$ bits, even if the leakage functions can only leak a single bit, it is possible to learn $k_n$ by having $l_i(k_i, R_i)$ leak the $i$th bit of $H^{n-i}(k_i)$. Since all these functions calculate $k_n$, it will be completely revealed. This class of attacks, while unrealistic in practice, are allowed within the theoretical model and must be protected against. They are known as future computation attacks [133]. Having extra values contribute to the key update function will not help as the leakage function has access to all inputs, and randomness cannot be used as it would mean that two parties cannot keep their keys in sync with ease. Hence, for security to be possible within this model an extra assumption will be required. The two major assumptions used within the literature are described below.

The continuous leakage model closely matches how hardware and attacks work in practice; each time a device performs another encryption (for example), the adversary can take another power trace as the device operates and this will give more information about the underlying secret key. By allowing the adversary to utilise arbitrary, adaptive functions it implies that a scheme proven secure in this model is secure against any side channels that behave in this way, even if they have yet to be discovered as an attack vector. The adaptivity means that the model also captures adversaries who try to and exploit multiple side channels at once, using one to aid the placement/measurement of the next.

### 3.6.1. Leak Free Components

Several works have solved the above issue using a leak free component [83, 97]. A leak free component is a part of the computation that the adversary is not allowed to leak upon. The goal is to make the leak free component as small as possible. If the whole scheme is contained
within the leak free component then the scheme is trivially secure against leakage. One standard approach is to only make the key update step a leak free component. This tends to either involve having a hash function, or something similar, which the adversary (or the leakage) does not have access to, used to update the key \([170]\) or the key is updated using some randomness and the leakage function is not allowed to leak on this randomness \([9, 61]\). Consider the future computation attack on the key update \(k_{i+1} = H(k_{i+1})\). If the hash function \(H\) is not allowed to be called by the leakage functions \(l_i\) then the future computation attack can no longer take place and it may be possible to prove the scheme, the key update is used within, secure within the continuous leakage model.

### 3.6.2. Only Computation Leaks Information (OCLI)

The OCLI assumption was introduced by Micali and Reyzin \([120]\) to attempt to add a theoretical underpinning to the way hardware behaves. This assumption is given in the form of the following 5 axioms:

1. Computation, and only computation, leaks information
2. The same computation leaks different information on difference machines
3. The information leaked depends on the chosen measurement
4. Information leakage is local
5. All the leaked information is efficiently computable from the computer’s internal configuration

Point one is the most important and thus becomes the most influential; if a device has two secrets \(k_1, k_2\) and during a computation only \(k_1\) is used in the computation (while \(k_2\) is stored in memory) then only \(k_1\) is passed into the leakage function and the leakage function does not have access to \(k_2\). Point four states that the leakage is local, meaning that the information leakage at a particular point in time is not influenced by any calculations performed, suitably far, in the past. Points two and three correspond to the fact that different computer and measurement set ups allow the adversary to extract different information (a smart card will leak differently to a server
Therefore, the adversary can adaptively choose the leakage function between calls. The final point relates to the behaviour of actual devices and thus the leakage functions chosen by the adversary must be restricted to be efficiently computable. From a theoretical point of view this is important, as this will stop the adversary from using the leakage functions to perform computations that are beyond his capability, such as solving discrete log problems.

This assumption, together with a certain class of schemes built from it, will be discussed in more detail in Ch. 5.

Pietrzak (and Dziembowski) [65, 133] took the OCLI assumption within the continuous leakage model and formalised it into what has now become known as the subfield of leakage resilient cryptography. This model will be made formal for the particular case of sharing a key in to two shares in Ch. 5 but an intuitive overview is given here. At the $i^{th}$ round, when the adversary makes an oracle call, they provide a leakage function $l_i$ such that the output of the leakage function is restricted to $\lambda$ bits. At each round the leakage function takes in any input from the previous round (if applicable), any randomness used within the round and the part of the secret state accessed this round. It is here that the OCLI assumption is used; if part of the secret state is not used within this round then it cannot be leaked upon. Finally, the leakage functions must be efficient, although it is possible to have a term representing their complexity within the advantage instead.

Pietrzak (and Dziembowski) [65, 133] construct leakage resilient PRGs. Since they are similar, the one by Pietrzak [133] will be given here as it is slightly simpler (given in Fig. 3.1).
scheme has two secret keys $k_0, k_1$ and an initial state $x_0$. On the $i^{th}$ call the adversary is given output $x_i$, which will also be fed into the next round. Since only one key is used per round, the adversary cannot leak on the other key. In this scheme a round is defined as a single call to the 2PRG $F$. Intuitively, security follows from the fact that each key is only used once by the 2PRG and then is updated, therefore the key cannot be completely leaked. The alternating structure using the new keys, together with the OCLI assumption, prevents future computation attacks and hence the scheme can be shown to be leakage resilient.

It is important to note that there are certain circumstances where the OCLI assumption does not hold. For example, it has been shown on certain devices, at the gate level, that independent computation does not necessarily lead to independent leakage [138]. A second example is when a device leaks Hamming distance, this gives information to the adversary about the item being computed on in comparison to the item previously stored (and no longer being computed on). Hamming distance leakage can be avoided by making sure that registers are zeroed directly after use. Therefore, for the majority of scenarios (including those we consider in this work) OCLI is a reasonable assumption to make.

3.7. Auxiliary Input

The auxiliary input model was considered in the context of leakage by Dodis et al. [54]. In this model, in addition to the standard information and oracle access given to an adversary, the adversary is also given auxiliary information $f(k)$ on the secret $k$, for arbitrary function $f$ providing that it is computationally hard to learn $k$ given $f(k)$. The model was originally used for studying hash functions [40] and obfuscation [80] with auxiliary input. The auxiliary input model is more general than the previous models described. In all the models described previously, the secret will maintain high min-entropy (in the other models if this was not the case the adversary could guess the key - for example if the adversary has leaked all bar 3 bits of the key they can just test the 8 remaining keys). In the auxiliary input model it is possible that $f(k)$ information theoretically determines the secret. Dodis et al. [54] give IND-CPA and IND-CCA symmetric encryption schemes within this model. The paper is concluded with a
discussion on how to enhance the model with additional adaptive leakage, instead of just the static leakage (the adversary was previously given \( f(k) \) before access to the oracle). The model is enhanced with a leakage oracle (the adversary still gets \( f(k) \), for hard to invert \( f \) at the start), for an arbitrary leakage function \( l \) which outputs a single bit and returns \( l(k, \cdot) \). The adversary is allowed to call the leakage oracle \( \lambda \) times, giving the adversary an extra \( \lambda \) bits of information about the secret key. In this model \( \lambda \) is a security parameter which must be less than the length of the key, otherwise the adversary can extract the whole key and the goal is to create a scheme which is secure even with the auxiliary input and the adaptive leakage.

In the asymmetric setting a slightly different notion is needed. Since there is a public key and a secret key the leakage function \( f \) will be of the form \( f(pk, sk) \). In the symmetric setting the assumption was essentially that it is hard to invert \( f(k) \) to learn \( k \). In the asymmetric case, if the security notion was still it is hard to invert \( f(pk, sk) \) then this would no longer be secure. It is possible to construct \( f \) such that it is a trapdoor function where \( pk \) is the trapdoor. Thus given \( f(pk, sk) \) it is computationally hard to invert but as soon as the adversary is given \( pk \) it becomes a trivial task. Hence the security statement has to be adjusted to state that given the pair \( (f(pk, sk), pk) \) it is hard to learn \( sk \).

The auxiliary input model does not directly correspond to the kinds of leakage an adversary will see in practice. However, the addition of the secondary adaptive leakage function brings the model closer to practice. This is mainly a theoretical model but has the advantage of capturing a wide range of leakage which are not included within any of the other models. Given this the auxiliary input model will not be considered further.

### 3.8. Relationship Between Models

In this section relationships between the previous models are discussed if such a relationship is known.
3.8.1. Bounded Retrieval Model and Bounded Leakage Model

In the bounded retrieval model the adversary is allowed to learn $\lambda$ bits of information about the secret state. This can be captured by the bounded leakage model because the leakage function will simply return the requested bits of the secret state (i.e. the leakage function will be the bit selector function). Thus, if a scheme is secure in the bounded leakage model against $\lambda$ bits of leakage it will also be secure in the bounded retrieval model against $\lambda$ bits of leakage.

It is important to note that the bounded retrieval model considers computationally unbounded adversaries while the bounded leakage model only considers bounded adversaries. Therefore, this reduction can only consider the computationally bounded case.

3.8.2. Continuous Leakage Model and Bounded Leakage Model

A scheme which is secure in the continuous leakage model that is allowed to leak $\lambda$ bits per ‘round’ will be secure in the bounded leakage model, against $\lambda$ bits of leakage, providing that all assumptions are still adhered to. If OCLI is being used, the only items being computed on are leaked upon and if leak free components are used then they are not leaked on. From this point it is reasonably easy to see that the implication is true, regardless of if the bounded adversary asks for all bits of leakage from a single ‘round’ or across multiple rounds, the leakage is within the bound for the continuous leakage model and thus is secure.

3.8.3. Probing Model and the Noisy Leakage Model

The relation between the probing model and the noisy leakage model have already been discussed within the probing model section. Any scheme which is secure in the probing model is also secure in the noisy leakage model. The number of probes $t$ a scheme is secure against is directly related to the noise within the noisy leakage model.
3.9. Variations of the Bounded and Continuous Leakage Models

As described above, and in a lot of the literature [9, 61, 75, 101], the way that leakage functions are modelled is that each time a leakage function is submitted, it may return at most $\lambda$ bits of leakage, where $\lambda$ is stated by the theorem. The disadvantage of this model is that it does not appear to match how DPA style attacks work in practice, since the bound $\lambda$ has to be less than the key size (normally a few hundred to a few thousand bits) while in a DPA attack the adversary will receive a trace which is typically several megabytes in size. In this section proposed variations to the description of the leakage function are given, together with how they relate to each other.

3.9.1. Bounded Leakage Functions

The bounded leakage function is the class of leakage functions described above, in which the output of the leakage function is allowed to be at most $\lambda$ bits.

An important question is how to capture the number of bits a device leaks per function call. This question must be answered so that practitioners can implement the schemes such that they meet the leakage bound. In Ch. 6 we relate the problem of bounding the leakage of a device to the key rank problem. The key rank problem states; given the output of a side channel attack and the secret key used within the device, if an adversary were to test all keys by decreasing likelihood, how many keys would they test before they tested the secret key. By performing a side channel attack and ranking the key with a varying number of traces, we can observe how the rank of the key increases with more traces. This can then be used as a metric to estimate how many bits of information were leaked per function call.

3.9.2. Entropy Preserving Leakage Functions

This class of leakage functions contains all leakage functions which reduce the HILL pseudoentropy [88] of the secret by, at most, $\lambda$ bits. HILL pseudoentropy is an analogue to min entropy where the distance is computational, instead of statistical (as all cryptographic adversaries tend to be computationally bounded). This can be seen as a more accurate representation of how
leakage works in reality. Consider a leakage function \( l(k) \) which returns \( \lambda \) bits of the secret \( k \) and compare it to the leakage function \( l'(k) = 0^n || l(k) \) where \( 0^n \) corresponds to a string of \( n \) 0’s (for fixed \( n \)) and || denotes concatenation. It is clear that both \( l \) and \( l' \) contain the same amount of information about \( k \). However, in the bounded leakage model \( l \) is a valid function while \( l' \) is not (for \( \lambda \) bits of leakage), yet when the class of leakage functions is the class which preserves HILL pseudoentropy, both of these leakage functions are valid. In the case of traces captured from a device, there will be a lot of redundant information that does not reveal anything about the key (such leakage on the program counter or leakage on any constants in the algorithm) and thus this class of leakage functions more closely model practice. Lots of work mention that security can be shown using this definition instead, with fairly trivial changes to the proof [101, 133].

### 3.9.3. Simulatable Leakage Functions

In this variation, the class of leakage functions are all functions that can be simulated. This variation is different from, yet complementary to, the simulatable leakage model which is described above. The simulatable leakage model is used to show that there exists a simulator for a given scheme which cannot be distinguished from real leakage, while this variation is showing security of a scheme in the continuous leakage model where, with each call, a leakage function is chosen from the set of simulatable leakage functions (in this scenario we have that for all leakage functions in the set there exists a simulator and not there exists a simulator for all leakage functions in the set). The leakage function is then evaluated on the secret and then returned to the adversary. The PRG given by Standaert et al. [158] can be seen to be within this model but where the leakage is restricted to the single simulatable leakage function. While it seems that if all leakage functions are simulatable then the scheme will be secure against continual leakage, this is not the case. The following example will give two leakage functions which are simulatable on their own but together completely reveal the key. Let \( F \) be a PRF then the functions \( l_1(k) = k + F_k(1), l_2(k) = 2 \cdot k + F_k(1) \) are simulatable since the adversary does not know \( F_k(1) \). Both these functions look random to the adversary and hence can be simulated. However, given both \( l_1(k), l_2(k) \) it is trivial to reveal the secret key \( k \). This shows that security within this model is non-trivial and still interesting to consider.
The strongly and weakly simulatable notions of Fuller and Hamlin [71] are also suitable within this context and in fact it is this situation they focus on within their paper.

### 3.9.4. Hard to Invert Leakage Functions

The class of leakage functions in this variation are all functions for which it is computationally hard to extract the secret information from the leakage. This can be seen to be extremely similar to the auxiliary input model. In the auxiliary input model the adversary is given the output of a single hard to invert leakage function before the security game begins. In contrast, in this variation of the continuous leakage model the adversary may adaptively send hard to invert leakage functions, each time an oracle call is made. A similar argument to the one given for simulatable leakage functions can be used to show that two functions which are hard to invert individually are not necessarily hard to invert when considered together.
3.9.5. Relationship Between the Various Leakage Function Classes

Fuller and Hamlin [71] discuss the relationships between various classes of leakage functions. For completeness this will be described in detail here. If each leakage class is considered as a set of leakage functions then it is possible to relate the leakage classes in terms of set containment. Figure 3.9.5 shows the relationships between the various leakage classes. Given results of the form \( A \subset B \), if a scheme \( X \) is secure in model \( M \) against leakage from \( B \) then \( X \) is also secure in the model \( M \) when the leakage is only from \( A \). Thus we can argue when a leakage class is more restrictive than others.

From Fig. 3.9.5 it can be seen that bounded leakage functions is the most restrictive form of leakage and thus gives the weakest security statements. This can be seen intuitively because it only gives a fixed number of bits of leakage regardless of the amount of information it contains. It has been shown previously, that bounded leakage functions are a subset of entropy preserving leakage functions. Therefore anything secure in model \( M \) against entropy preserving leakage functions is also secure in model \( M \) against bounded leakage functions. The converse is not true. However, multiple works [101, 133] argue why it is trivial to convert their proof against bounded leakage functions to a proof against entropy preserving leakage functions. Note that this is not the same as showing their equivalence but instead the argument is being made that, for a particular scheme, it may also be shown to be secure against entropy preserving leakage functions using the same techniques.

While it is clear from Fig. 3.9.5 that the class of hard to invert leakage functions is the least restrictive, and thus the most desirable, the majority of work has not been in this variation of the model. This is due to the fact that the assumption is so weak that the leakage functions can be used to receive a lot of extra information. For example, for a message authentication scheme a message/tag pair is hard to invert to retrieve the key (otherwise it would be easy to create a forgery given a single tag/message pair) yet this provides a forgery to the MAC EUF-CMA game. While from a theoretical perspective it is the most desirable form of leakage, considerations must be made into the practicality of the resulting schemes. The goal of this thesis is produce schemes which are secure against as wide a range of leakage function as possible while still being practical.
Other results of note are that there are leakage functions which only reveal a bounded number of bits which are not simulatable and that there are hard to invert leakage functions which are also not simulatable.

### 3.9.6. Non-Adaptive Leakage Functions

A weaker variation of all the options presented above is when the leakage functions have to be chosen by the adversary non-adaptively before the game begins. Therefore, the adversary cannot use information learnt from either the leakage or from the output of oracle to inform the next choice of leakage function. Regardless of the class of leakage functions (e.g. simulatable or bounded) non-adaptive leakage is almost always weaker than adaptive leakage.

### 3.10. Conclusion

As shown above, there are a multitude of different leakage models available within the literature. In this thesis the aim is to ‘bridge the gap’ between theory and practice. The goal is to create schemes that are theoretically secure, in a model which relates to how attacks work in practice, while being efficient enough to implement and use. Some models are too restrictive (for example the AC⁰ leakage does not allow Hamming weight to be calculated, which is how devices tend to leak in practice [110]), while other models can be so general that the resulting schemes are not practical, such as the auxiliary input model. Therefore, for the remainder of the work we focus on two leakage models.

The first model considered is the simulatable leakage model. Following the design paradigm of Standaert et al. [158] we will be considering simulators that can be instantiated on a real device. This implies that all leakage considered will be real leakage from actual devices that require protection. The advantage is that since the leakage considered relates to the inherent properties of a physical device, a mathematical leakage function does not have to be defined. Therefore the leakage can be neither too weak or too strong; it will be practically relevant and accurate by design. One disadvantage of this methodology is that the simulator would have to be tested for each different implementation, on each device to check that the assumption still
holds, as the simulators can be device/implementation specific (however the goal is to design a universal simulator that works for the given leakage regardless of device or implementation). Another disadvantage is the simulator only works for the designed leakage function and if a new attack vector is discovered, a new simulator would be required. While the assumption of the existence simulator allows schemes to be shown to be theoretically secure, as we touch on in Ch. 4, it is not apparent how to theoretically prove the security of a simulator. Therefore, while this model allows for some of the most efficient schemes to be constructed, there is a trade off of how much can be shown to be theoretically secure and how much must be practically verified. However, we feel that the benefits of this model outweigh the disadvantages and the theoretical vs practical tradeoff of this model should be investigated further.

The second model considered is the continuous leakage model, as it closely represents practice. The fact extra leakage is allowed per query corresponds with the fact traces can be captured each time a device is used. The fact that leakage functions can be chosen adaptively between calls corresponds to adjusting the measurement set up between rounds. While the leakage functions allow attacks that could not occur in practice, such as future computation attacks, it does capture the majority of the attacks used in practice and the leakage is not so powerful that it makes the resulting schemes non-practical (it has been argued that it does not capture some forms of cold boot attacks [87]). Between using the OCLI assumption and a leak-free component, we have chosen to use the OCLI assumption since we feel that this models practice more closely. It is possible to construct a device where the computations are suitably separated in time such that their leakage will be independent, while it seems incredibly difficult to construct a component that does not leak at all. Since the goal of this work is to bridge the gap between theory and practice, this model was chosen as it will allow a host of efficient schemes to be proven secure against a wide range of practical attacks.

While not chosen for further consideration within this work, the probing model, especially its relation to the noisy leakage model, is an extremely desirable model. Giving an adversary access to the internal variables of the scheme after they have had noise added to them, closely models what an adversary sees from a wide range of attacks, such as power or EM attacks. However, it is important to note that it does not cover all attacks, e.g. timing attacks. The model also does not
capture attacks which are just a relic of the model, such as future computation attacks. Another advantage of this model is that it allows the construction of some extremely efficient schemes. The reason we did not choose this model for our work is that at the time of choosing almost all schemes within this model relied on some form of leak free component. It was not apparent at the time of choosing the model if this leak free component was inherently required. Due to the unrealistic nature of having a component, we did not consider the noisy leakage model any further. However, as mentioned previously, there has since been work within the model that does not require the use of a leak free component [17, 42, 47]. Therefore, this model, alongside the continuous leakage model, offers an interesting trade-off between the efficiency of the schemes and the range of leakage captured by the model.
4. Simulatable Leakage Model

The work carried out in this chapter was joint work with Mr Jake Longo Galea, Professor Elisa-beth Oswald, Dr Daniel Page, Dr Martijn Stam and Dr Mike Tunstall and appeared at Asiacrypt 2014 [108]. My contribution in this work was in the theoretical modelling side, designing the theoretical simulator that meets the enhanced security definition and advising with the implementation where required. All data used within this chapter (power traces etc) is from the original paper and were captured from devices by Jake Longo Galea, who implemented the simulators in hardware. Any figures relating to power traces were drawn by Jake Longo Galea, while any figures for the theoretical modelling and simulators were created by myself. Two parts of the paper have been excluded from the thesis. The first is the instantiation of the simulator on multiple devices because this was performed by Jake and the results confirm what a single example (shown in this chapter) provides. The second is various attempts at making a practical simulator. These were designed by myself and Jake but were not successful (can be attacked in a similar way to the original simulator) and thus would not add anything to this chapter.

In this chapter we first recap on the relevant previous work [158] that has been presented for simulatable leakage, before explaining why the current model does not accurately represent how a device or adversary would behave in the real world. Given this insight, we propose a new model which better represents how a device and adversary behave. Our model is a generalisation of the original model and thus anything secure in the new model will be secure in the original model. We then show that the solution provided when the model was originally defined is not secure. We do this by constructing a distinguisher, who can win the new security game. The chapter is concluded with a theoretical simulator which can be shown to be secure against our distinguisher (it remains an open question if the simulator is secure against all distinguishers) and then we
4.1. Previous Work in the Simulatable Leakage Model

The simulatable leakage model’s goal is the same as other leakage models; to be able to prove security of schemes even when certain leakage is involved. Other models achieve this by defining a class of leakage functions and showing security against this class of leakage functions. The simulatable leakage model differs in that it does not mathematically define a class of leakage functions but argues that if the leakage can be simulated, without the knowledge of the key, then the leakage cannot give the adversary any additional information. The main piece of work in this area defines a security notion for block ciphers [158]. They then go on to construct a simulator for AES against DPA style attacks. The reason AES was chosen is due to its widespread use. By being able to create a leakage simulator (for power traces) for AES it would show that the leakage for any current implementation of AES, which meets the requirements, would not be exploitable (using power measurements). The simulator would have to be checked on a per device basis, as each device leaks differently. While there is other work within the model [71, 131, 132], it is not the main focus of this work and thus it will not be discussed until the further work section where it will be discussed in more detail.

4.1.1. The Model

The high level intuition of the simulatable leakage model is as follows; if it is possible to construct a simulator for the leakage (which does not have access to the secret state), such that any adversary cannot distinguish between real leakage and simulated leakage, then the leakage must contain no information about the secret key. Hence the security of the underlying scheme with leakage will be the same as the scheme without leakage.

This way of modelling leakage has several advantages. Firstly, it finds a suitable balance between theory and practice, something which has been difficult to achieve for schemes involving leakage. Schemes have tended to be either provably secure and either inefficient or never implemented (e.g. some of the leakage resilient schemes [65]), or they are efficient and have
been implemented but come with no proof of security (such as certain masking schemes [110]). The existence of a simulator for an underlying primitive allows theoreticians to build more complex, but still provably secure, primitives on top of the underlying primitive [131, 158]. From a practical point of view, if the simulators are efficient it is possible to instantiate these games in hardware and possibly falsify the indistinguishability properties of the simulator for the particular hardware (the original paper talks about verifying that the simulator is indistinguishable from a real trace. This would involve showing indistinguishability against all distinguishers. In practice an evaluation lab would falsify the claim by finding attacks against the simulator). Since each device has different physical characteristics, the simulatability will have to be checked on a per device basis but still allows a practitioner to check that a scheme behaves, in practice, in a manner that agrees with the theoretical statements. To prove security against leakage, the security must be shown for a certain class of leakage functions. Leakage must be restricted to come from a certain class, as for some leakage functions it will be impossible to prove security (such as the function which returns the key). In the literature, defined classes of leakage functions have either been too weak (i.e. $\text{AC}^0$ [68]), too strong (i.e. any polynomial time function with bounded output [65]) or have not captured how a device behaves in practice (i.e. any hard to invert function [54]). The advantage of the simulatable leakage model is that no such class of leakage needs to be defined, as a simulator will be created for the physical side channel security is required for (such as power). The downside of this is that a simulator is required per leakage function to be analysed. As per Standaert et al. [158] we will be considering power traces for the remainder of the chapter.

Following the work of Standaert et al. [158], we only consider constructing a simulator for a block cipher in this work, as well as how to use the block cipher to construct a PRG which is secure in the face of leakage. When a concrete instantiation of a block cipher is required we will use AES-128.

**Notation.** The probabilistic leakage of a block cipher $\text{BC}$ will be denoted $\text{BC}_k(x) \sim \Lambda \overset{\text{def}}{=} l(k, x)$ where $l$ is a leakage function, $x$ is the plaintext and $k$ is the secret key. The leakage function can be described as a vector $l = (l_1, \ldots)$. For block ciphers, which usually consist of
experiment \( \text{Exp}^{q\text{-sim}}_{BC}(A, l, S^l, b) \):

\[
k, k^* \overset{\$}{\leftarrow} K
\]

\[
(i, j, f) \leftarrow (0, 0, 0)
\]

\[
b' \leftarrow A^{\text{Enc}(/),\text{Gen}(/),\text{Leak}(/)}(i, j, f)
\]

Return \( b' \)

\[
\text{proc Enc}(x):
\]

\[
i \leftarrow i + 1
\]

if \( i > q \) then

\[
\text{Return } \$
\]

end if

\[
c \leftarrow B_{C_k}(x)
\]

if \( b = 0 \) then

\[
\Lambda \leftarrow l(k, x)
\]

else

\[
\Lambda \leftarrow S^l(k^*, x, c)
\]

end if

Return \((c, \Lambda)\)

\[
\text{proc Gen}(z, x):
\]

\[
\text{if } f = 1 \text{ then}
\]

\[
\text{Return } \$
\]

end if

\[
f \leftarrow 1
\]

if \( b = 0 \) then

\[
\Lambda \leftarrow S^l(z, x, k)
\]

else

\[
\Lambda \leftarrow S^l(z, x, k^*)
\]

end if

Return \( \Lambda \)

\[
\text{proc Leak}(z, x):
\]

\[
j \leftarrow j + 1
\]

if \( j > s_A \) then

\[
\text{Return } \$
\]

end if

\[
\Lambda \leftarrow l(z, x)
\]

Return \( \Lambda \)

Figure 4.1: \( q \)-simulatable leakage

several encryption rounds, these points can be grouped such that each group relates to a round of the block cipher. This will be denoted using a superscript. For example, for AES-128 the leakage can be grouped by the ten rounds and denoted \( l = [l^1, \ldots, l^{10}] \). When a subset of the rounds is required, \( l^{i,j} \) will be used to denote \([l^i, \ldots, l^j] \). Concatenation will be denoted by \( || \). When sets of leakage are required (such as multiple traces), it will be denoted with a bold typeface. \( l^{i,j} = [l^i, \ldots, l^j] \) corresponds to taking the leakage for rounds \( i \) through \( j \) for all leakage in the set. Finally, when points within a set are required to be indexed directly subscript will be used, \( i.e., l_{u} \) refers to taking the \( u \)th point in the leakage vector \( l \).

With the notation in place, the formal definition of the model is denoted as the \( q \)-simulatable leakage security notion [158] and is defined as follows.

**Definition 28** (\( q \)-simulatable leakage \((q\text{-sim}) \) [158]). Let \( BC \) be a block cipher. Then Fig. 4.1 defines the \( q \)-sim security game. The advantage of an adversary \( A \) winning the game is defined as

\[
\text{Adv}^{q\text{-sim}}_{BC,l,S^l}(A) = \Pr[\text{Exp}^{q\text{-sim}}_{BC}(A, l, S^l, 1) = 1] - \Pr[\text{Exp}^{q\text{-sim}}_{BC}(A, l, S^l, 0) = 1].
\]

The experiment begins by choosing two independent keys, one to be used by the block cipher,
while the other key is given to the simulator. The adversary is then given access to three oracles. The adversary can call the $\text{Enc}$ function at most $q$ times and will receive back the encryption of the message $x$ under the real key $k$ and some leakage. Depending on the bit $b$, the adversary will either receive the real leakage $l(k, x)$ on the key $k$ and message $x$ used to calculate the returned ciphertext $c$ or they receive simulated leakage $S(l)(k^*, x, c)$ where the simulator takes in the message $x$, the ciphertext $c$ corresponding to $x$ under key $k$ and an independent key $k^*$. The superscript $l$ on the simulator denotes that the simulator can directly call the leakage function $l$. The function $\text{Leak}$ can be called $s_A$ (for some parameter $s_A$) times and allows the adversary to provide a key and message and learn the real leakage on these values. This oracle represents an adversary’s ability to profile a device, i.e., attempt to derive or represent information about the leakage function. Note that the adversary can query the leakage on the same plaintexts that will be used as challenges to the $\text{Enc}$ oracle, so specific templates for the inputs used as challenges can be derived. The final oracle is the $\text{Gen}$ oracle which represents the fact that encryption keys (especially when the key is updated) can be derived using the previous key. The $\text{Gen}$ oracle can be called only once. The oracle takes in a message $x$ and key $z$ and returns simulated leakage using either $k$ or $k^*$ as the ciphertext, depending on the value of $b$. The simulator does not require that the plaintext/ciphertext pair $(x, c)$ is a valid pair under the key $k$. It is possible the simulator may not have a way to verify $(x, c)$ form a valid pair, as the simulator does not have access to $k$. The $\text{Gen}$ oracle allows the adversary to exploit the leakage of the key being generated, instead of just the leakage from the key being used.

We try to explain some of the design decisions made when creating this security notion, before stating how the model was used to create a simulator and build a secure PRG upon it. The $\text{Gen}$ oracle is very specific to the goal of creating a PRG. The majority of uses of block ciphers do not have the key as an output from a previous block cipher call, where as this is the case in some PRG constructions. Standaert et al. have chosen to lose some generality from their security game in order to give the adversary more power for the chosen scenario. It is unusual that the $\text{Gen}$ oracle calls the simulator in both the real and simulated worlds. In most security experiments the simulator is never called in the real world. It is not apparent to us why this design decision was made but since all of our attacks do not require use of the $\text{Gen}$ oracle we do not investigate
it further here. The \( q \)-sim game has \( q \) as a clear parameter in the definitions but \( s_A \) is not. This is because in a practical side channel attack the restriction tends to be how many traces can be collected from the device being attacked. Templates (represented by the Leak oracle) tend to be easier to acquire as they can be captured from the adversary’s own copy of the device. The other interesting design decision is the inputs \( k^*, x, c \) to the simulator. It is apparent that the simulator must be given the input \( x \) to be able to simulate leakage on the encryption of \( x \). The simulator is given \( c \) because in practice an adversary can perform a DPA attack on AES using the plaintext \( x \) on the first few rounds or the adversary can attack the last few rounds using \( c \). The simulator is given a key \( k^* \) to help with consistency between simulated traces if required. The final simulator we give in this work does not require the use of a key \( k^* \) (consistent between calls) and thus it will be excluded. It would be possible to generalise the definition such that the simulator is passed in a state instead of a key.

4.1.2. The Simulator

After defining the \( q \)-sim security game for block ciphers, Standaert et al. [158] propose a leakage simulator for AES-128, when the leakage is a power trace. The advantage of constructing a simulator for AES is its widespread use. By constructing a simulator for AES, instead of a custom block cipher, each device using AES can possibly achieve security in the face of leakage. To provide security the device must be checked to verify that the simulator cannot be distinguished from a real trace. This must be done per device. However, checking each device is a lot more desirable than having to change the implementations which are already used in practice.

The split and concatenate simulator is given in Fig. 4.2 and takes in a plaintext/ciphertext pair \((x, c)\) and a key \( k^* \). Note that \( k^* \) is independent of the key used to create the plaintext/ciphertext pair. The simulator first encrypts \( x \) under \( k^* \) and records the power trace. It then decrypts \( c \)
under $k^*$ to get $x'$ and encrypts this value under $k^*$ so that the power trace can be recorded. It is worth noting that $x'$ will encrypt to $c$ under $k^*$ by design. The leakage is captured on the second encryption and not the decryption because in certain devices encryption and decryption may leak differently. This would trivially make the simulator distinguishable. The two leakage traces are then split in half and the first half of the trace corresponding to encrypting $x$, is concatenated with the second half of the trace corresponding to encrypting $x'$, and this is then returned as the simulated leakage trace.

The intuition as to why this simulator works is as follows. In standard DPA attacks, the adversary attacks either the first few rounds or the last few rounds. This is because of the diffusion properties of AES. To perform a DPA attack the adversary must be able to model some intermediate computation $f(K, M)$ and past the first few rounds there would be too many unknowns. See Sect. 1.1.1 for more details. The simulated trace is designed to reflect the fact the adversary can only exploit the first few or last few rounds; the start of the trace will relate to being an encryption of $x$, while the end of the trace corresponds to a message which encrypts to $c$. Since DPA attacks do not exploit the middle rounds, this seems a feasible place to make the switch between the two traces. The implicit assumption that the key cannot be recovered must be made. If the adversary can recover the key from the set of traces they will be able to distinguish the real and simulated traces by checking that $x$ encrypts to $c$ under the recovered key. In the scenario the simulator is used within, $q$ will be suitably small such that this assumption holds in practice on the majority of devices, for DPA style attacks. However, for SPA attacks which use one or few traces, such as algebraic attacks, this assumption is violated. Therefore, the model does not capture these forms of attacks and hence the model assumes that the adversary is unable to perform such an attack, as these attacks would allow the adversary to extract the key and win the game.

4.1.3. The Construction

Within this model, and with a simulator in place, Standaert et al. [158] construct a PRG $G$ which is secure against attacks where the adversary also receives leakage in the form of power traces. The PRG is given in Fig. 4.3 and is reasonably ‘standard’ and widely used [99]. Having
a proof of security in the face of leakage for the PRG implies that any device that meets the 2-
simulatability requirement inherits the provable guarantees without having to change their con-
struction. It is important to note that each device will have to have its 2-simulatability checked
individually for each leakage function security is required against (we only consider power traces
within this work).

The PRG works by calling a 2PRG each time new output is required. The 2PRG takes in a
key and creates an output of double the length; half of which is used as output, while the other
half will be used as the key for the next round. The 2PRG is constructed by making two calls
to the block cipher $BC$ where the input to the 2PRG is used as the key to encrypt two known
plaintexts and the two ciphertexts are output by the 2PRG.

Based on the given construction, it is fairly straightforward to see that the proof of security
only requires that $q = 2$, since each key is only used twice before it is updated: once to encrypt
a public value ($p_0$) to construct the output and once to encrypt a different public value ($p_1$) to
update the key. The two public values ($p_0, p_1$) correspond to the $q = 2$ queries an adversary
would receive from the Enc oracle for this construction. The Gen oracle is used to capture the
leakage when the key was calculated in the previous round. Below, the theorem statement for
the PRG is recapped.

**Theorem 1** ([158, Theorem 1]). Let $BC : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^n$ be a block cipher and $l$ be a leakage function. Let $A$ be an adversary against the lr-PRG security of $G$, then there exists two adversaries $B, C$ (of similar complexity to $A$) against the PRF security of $BC$ and the 2-simulatability of $BC$ with leakage function $l$ and simulator $S^l$ such that:

$$
Adv_G^{lr-prg}(A) \leq 2 \cdot q \cdot (Adv_{BC}^{prf}(B) + Adv_{BC,l,S^l}^{2-sim}(C))
$$
where $q$ is the number of PRG queries made.

The lr-PRG security game is defined, similarly to the PRG security game without leakage (see Def. 13) except that for each call to $F_n$ the adversary also receives the leakage function $l$ applied to the current state $s_i$. For reference, the PRF security game can be found in Def. 11.

4.2. The New Simulatable Leakage Model

In this section we argue why the model given by Standaert et al. [158] does not fully capture how either a real device or an adversary would behave in practice. Recall that the model only allows leakage on one PRG state and, even if used in a construction (such as the one given in Fig 4.3), only one of these PRG states would leak. This is depicted in Fig. 4.4. However, realistically, an adversary has access to multiple calls of the 2PRG (it would be extremely unusual to only have access to a single call of a PRG). Therefore they would try to take advantage of all information available. It is unlikely to be able to construct a device which only leaks on a single 2PRG call. The majority of devices would use the same piece of code/circuit for the 2PRG and just update the key being used. Thus all 2PRG calls should leak in the same manner.

In reality, an adversary would take advantage of leakage from all PRG calls, as depicted in Fig. 4.5. Below we formally define the $p$-$q$-simulatable leakage security to capture how an
adversary would behave in practice.

**Definition 29** (*p*-*q*-simulatable leakage (*p*-*q*-sim)). Let $BC$ be a block cipher. Then Fig. 4.6 defines the *p*-*q*-sim security game. The advantage of an adversary $A$ winning the game is defined as

$$\text{Adv}_{(BC,l,S^l)}^{p-q\text{-sim}}(A) = \Pr[\text{Exp}_{BC}^{p-q\text{-sim}}(A, l, S^l, 1) = 1] - \Pr[\text{Exp}_{BC}^{p-q\text{-sim}}(A, l, S^l, 0) = 1].$$

The security game here behaves similarly to the *q*-sim game except the adversary is playing a game with *p* distinct keys, instead of a single key. For each oracle the adversary now passes in an index of which round key they want the oracle to use. We opted for a single simulator to be used for creating the *q* leakage traces for each of the *p* PRG states. We chose to use a single simulator because the simulator has no knowledge of the secret state (the secret state of the PRG is used as a key for the block cipher and hence the switch between naming depending on context) but is designed for a particular leakage function. In the *p*-*q*-sim game we assume that each PRG state leaks using the same leakage function. It would be possible to generalise this model such that a different leakage function is used per PRG state but we do not investigate this here. This generalisation falls into the continuous leakage model where the leakage functions are taken from a certain set of simulatable functions. This was discussed in more detail in Sect. 3.9.

The other important thing to note is that the leakage is either all real or all simulated across all PRG states and is not different per PRG state. In an evaluation context, which is discussed by
experiment $\text{Exp}_{\text{BC}}^{p-q\text{-sim}}(A, l, S^l, b)$:
\[
\{k_m\}_{m=1}^p, \{k^*_m\}_{m=1}^p \leftarrow K
\]
\[
\{(i_m, j_m, f_m)\}_{m=1}^p \leftarrow (0, 0, 0)
\]
\[
b' \leftarrow A^{\text{Enc}(:), \text{Gen}(:), \text{Leak}(:)}()
\]
Return $b'$

proc $\text{Enc}(x, m)$:
\[
i_m \leftarrow i_m + 1
\]
if $i_m > q$ then
\[
\text{Return } \odot
\]
end if
\[
c \leftarrow \text{BC}_{k_m}(x)
\]
if $b = 0$ then
\[
\Lambda \leftarrow l(k_m, x)
\]
else
\[
\Lambda \leftarrow S^l(k^*_m, x, c)
\]
end if
Return $(c, \Lambda)$

proc $\text{Gen}(z, x, m)$:
if $f_m = 1$ then
\[
\text{Return } \odot
\]
end if
\[
f_m \leftarrow 1
\]
if $b = 0$ then
\[
\Lambda \leftarrow S^l(z, x, k_m)
\]
else
\[
\Lambda \leftarrow S^l(z, x, k^*_m)
\]
end if
Return $\Lambda$

proc $\text{Leak}(z, x, m)$:
\[
j_m \leftarrow j_m + 1
\]
if $j_m > s^A$ then
\[
\text{Return } \odot
\]
end if
\[
\Lambda \leftarrow l(z, x)
\]
Return $\Lambda$

Figure 4.6: $p$-$q$-simulatable leakage

Standaert et al. [158], this is realistically the game that would be used.

We elaborate on some of our design decisions: Firstly, both the real keys and the keys that are used by the simulator are chosen independently – the PRG states are not generated in the manner designated by the scheme. This helps to keep the security notion scheme independent. An adversary who can win the $p$-$q$-sim game would be able to win the game where the keys are generated according to the scheme, since if the adversary can win without exploiting the extra structure they will be able to win when utilising the extra structure that the key update provides. Secondly, the Leak oracle takes an index $1 \leq m \leq p$ (in the other oracles this corresponds to which of the $p$ states to use) but Leak does not have access to any states, so this does not effect the oracle and thus is not strictly necessary. It would be possible to simplify the oracle, such that instead of giving the adversary $p$ sets of $s^A$ queries, the adversary is just given $p \cdot s^A$ queries. However, by describing the game in this manner, it is slightly more complex but it allows for clearer relations between the $q$-sim and $p$-$q$-sim games to be drawn. Finally, while in the given PRG, the adversary would have access to the PRG states sequentially, the game allows
the adversary to access the states in an arbitrary order. If the scheme is secure in this model it will also be secure in the model where the PRG state order is enforced.

The \( p-q \)-sim game is a generalisation of the \( q \)-sim game, if \( p \) is set to one, the \( p-q \)-sim game is identical to the \( q \)-sim game. For reasons discussed above, not only is it a generalisation of the \( q \)-sim security game but it also more accurately models how a device and an adversary would behave in practice. It would be tempting to believe that an adversary cannot exploit the ability to have access to extra PRG state leakage. Since standard DPA style attacks make a key hypothesis, each time the key is refreshed the adversary must start a new attack without any prior information. Standaert et al. [158] argue why the game, in general, does not hybridise over \( q \) but it is not discussed if the game were to hybridise over \( p \). A game is hybridisable if it can be split in to a series of smaller ‘hybrid’ games such that security of all the individual games implies security of the entire scheme. Taking hybridising over \( q \) as an example, the \( i \)th hybrid game would return real traces for the first \( i \) queries to \( \text{Enc} \) and simulated traces for queries \( i \) through \( q \). The hybrid with \( i = 0 \) corresponds to the simulated world, while when \( i = q \) the hybrid corresponds to the real world of the \( q \)-sim game. If the game were to hybridise over \( q \), if an adversary could not distinguish between hybrids \( i \) and \( i+1 \) they would not be able to distinguish between the real and simulated worlds of the \( q \)-sim game. The \( p-q \)-sim game hybridising over \( p \) would imply that \( \text{Adv}_{\text{BC},l,S^i}^{p-q-sim}(A) \leq p \cdot \text{Adv}_{\text{BC},l,S^i}^{q-sim}(B) \). In the next section we construct a distinguisher that can work across different keys, thus utilising the full power of the more realistic \( p-q \)-sim security notion. The reason that the game does not hybridise over \( q \) can be seen with the following example. Consider a key that is shared in to \( d \) shares and the leakage function gives the adversary a single share (of their choice). Then with \( d-1 \) shares the adversary has no information about the key while with \( d \) shares the key is completely revealed and thus neither game hybridises in \( q \). We leave it as an open question as to whether the \( p-q \)-sim game hybridises over \( p \).
4.3. Breaking the Simulator and the Challenges of Fixing it

This section begins with a discussion of cross-correlation and how it can be used as a distinguisher. Next, it is shown that a cross-correlation distinguisher can differentiate between real traces and ones created by the split and concatenate simulator. The section is concluded with a brief discussion of methods to try and protect this simulator or variations of the simulator such that it produces traces indistinguishable from real traces. Unfortunately we were unable to protect the simulator, and hence the requirement of the theoretical simulator in the next section.

Whilst the proposed simulator can be instantiated with any invertible function, the construction of the PRG requires a PRF. A block cipher meets the given requirements. We continue to follow Standaert et al.’s exposition and use an implementation of AES-128 as the running example. Not only does this continue in the path of previous research, it is also the obvious choice. Since AES is the most widely spread block cipher, a simulator here would imply a simulator for a great deal of devices in practice. Before we construct the distinguisher we recap how leakage behaves in a real world device and how it relates to information flow within AES.

4.3.1. Properties of Real World Leakage

All high level functions are processed at the gate level when implemented and a clock controls when data flows between gates. At each clock cycle, each change in the state of a gate produces leakage (such as power consumed or EM emitted) which can be captured by an external attack through monitoring the device. Figure 4.7 shows the leakage, in the form of power consumption,
from an AES implementation on an 8-bit microcontroller (the AES implementation operates on bytes). Figure 4.7(a) shows the power trace for a single round. In the full trace a single round can be detected by looking for the repeating pattern corresponding to the AES rounds.

For simple architectures, it is possible to see how individual instructions affect the power consumption of a device. Figure 4.7(b) shows a zoomed in portion of the trace corresponding to the SubBytes operation. The power variations corresponding to register transfers (MOV, MOVC), an increment (INC) and a conditional branch (DUNZ), to loop over all 16 of the state bytes, can be seen within the trace.

If a parallel implementation is considered, multiple state bytes can be operated on at once. However, to maintain correctness, certain operations must be performed sequentially. For example, in AES all 16 SBox lookups in a round may be performed in parallel but the AddRoundKey must occur before SBox lookups to maintain the correctness of the device. It is possible to implement the SBox as a combinatorial circuit instead of as a lookup table [168]. A combinatorial circuit is not controlled by a clock but the output of the circuit tends to be connected to some form of synchronous storage element. Figure 4.8(c) shows a power trace for a device implemented in this manner and 10 visible peaks can be seen relating to the 10 rounds in AES (the 11th peak relates to the writing out of the result).

4.3.2. Cross-Correlation as a Distinguisher

Correlation relates to a wide class of statistical dependencies within data. There are a multitude of different techniques which can be used to capture information about the correlation of data. Correlation has been used within side channel analysis previously [37] and its capabilities as a distinguisher are reasonably well understood [111]. In the context of side channel attacks, the Pearson correlation coefficient tends to be the preferred correlation metric and thus will be used for the remainder of this chapter. For completeness it is recalled below.
The Pearson correlation coefficient is a measure of the linear correlation between the two inputs. The output is a value between -1 and 1, where 0 is no correlation, -1 is perfect negative correlation (as one variable increases the other decreases and vice-versa) and 1 is perfect positive correlation (as one variable increases, so does the other). It is calculated by computing the covariance of the two variables (a measure of how much the two variable change together) and normalising by their standard deviations.

Cross-correlation is a term used within signal processing and is used to measure the similarities of waveforms. Cross-correlation takes the two signals and calculates the correlation between them at each step, as one signal is moved ‘in time’ with respect to the other signal. The cross-correlation has also been used within the context of side channel analysis [119, 144, 167].

**Construction of Cross-Correlation Traces**

Any implementation of AES (or any other algorithm) will operate on the intermediate values in a serial manner. Hence intermediate values at a particular point in time have extremely high correlation to related intermediate values. For example, the input to ShiftRows will be highly correlated to both the output of the SubBytes step (the values are the same) and the input of the MixColumns operation. As an effect of this, and because leakage is related to the data that a device is computing on, the leakage at these points will also have a high correlation (see [110, Ch 4]). High correlation is also expected between program data that is independent of the key and states, e.g. the program counter, memory location pointer, etc. Therefore, any implementation will have a unique cross-correlation ‘fingerprint’.

By producing a cross-correlation matrix that shows the cross-correlation of all pairs of points \( \{ \rho(l_u, l_v) \forall u, v \} \), it is possible to capture all correlation relationships represented within the leak-
age trace. However, this will result in a cross-correlation matrix which has size quadratic in the size of the leakage trace which is incredibly large. For example, a typical, filtered, trace for the 8051 contained approximately 100,000 points and was approximately 0.8MB in size, thus the cross-correlation trace would contain 10,000,000,000 points and would be approximately 7.5GB. We therefore decided to only store the maximum (absolute) correlation value for each point in the trace \( \tilde{\rho} = \{ \tilde{\rho}_v \forall v \} \) where \( \tilde{\rho}_v = \max \{ \rho(l_u, l_v) \forall u \} \). To avoid the trivial correlation for each point, points within a small ‘window’, around the point being considered, are ignored. This avoids the trivial correlation with itself and correlation with leakage points for the same operation where the operation spans several clock cycles. This window size is device dependent and was empirically calculated for the device we worked with. The resulting cross-correlation trace will therefore be the same size as the original leakage trace. This will allow consistency to be tracked within the trace and any ‘dip’ in the expected correlation will allow discontinuity of the underlying data to be detected. To make the cross-correlation more efficient we only looked in one direction through the trace (e.g. \( \forall u : u < v \)) as this helped to speed up the cross-correlation calculation without having any detrimental effect on the results. Since less information is being stored than the full cross-correlation, this can only weaken the adversary and therefore, success in this scenario would imply success of the more powerful adversary who has complete knowledge of all cross-correlation values.

Figure 4.8 provides an illustrative example of two devices’ power traces together with their corresponding cross-correlation traces (calculated using multiple power traces). For contrast, we chose two devices that operate in different manners to demonstrate that there will be correlation within a power trace regardless of the underlying architecture. The first device is the 8051 and implements a ‘standard’ serial 8-bit implementation of AES. The 8051 is highly representative of low cost devices that are available in the market today and in widespread use. The second device is the SASEBO-R and implements a highly parallel version of AES. This style of device is more suited to high end products where security and speed are two important factors.
4.3.3. Distinguishing Between the Simulator and Real Leakage

In both the $p$-$q$-sim and $q$-sim games the distinguisher has access to three oracles $\text{Enc}$, $\text{Gen}$, $\text{Leak}$. The cross-correlation distinguisher will only use the $\text{Enc}$ oracle when trying to differentiate real traces from simulated traces. It is clear that this oracle has to be called to receive the challenges, else the adversary cannot win with probability better than guessing. The oracle will be called $q$ times to maximise the chance of success. When $q$ is sufficiently small (such as in the scheme given where $q = 2$) more traces may be required to accurately calculate the cross-correlation trace. The more general $p$-$q$-sim game can be used in this scenario, as the adversary has access to $p \cdot q$ traces. In a practical setting $p$ would have to be suitably large, so that a device could be repeatedly used without having to be replaced. This implies an adversary has access to a lot more traces to calculate the cross-correlation trace. In some cases, the $\text{Leak}$ oracle may also be used to template a device by calculating cross-correlation traces on the real leakage for adversarially chosen keys. These traces will then be able to be compared to the cross-correlation traces produced from the challenge leakage. The $\text{Gen}$ oracle is not required at any stage by our distinguisher.
Since there is a single simulator for all adversaries (instead of a simulator per adversary), we assume that the adversary’s knowledge includes the implementation details of AES (e.g. combinatorial SBox vs lookup table) as well as the behaviour of the simulator, i.e. where the traces are split and concatenated.

The distinguisher works against the split and concatenate simulator by applying the cross-correlation method described previously to a set of $q$ traces (it works similarly in the $p$-$q$-$\text{sim}$ game). Recall that the cross-correlation trace shows ‘patterns’ based on other intermediate values related to the given points. The simulator will (mainly) show the same level of correlation because it is using real traces and their intermediate values correlate with other intermediate values. However, the point in the simulated trace just after the concatenation point (the adversary is assumed to know this point) will no longer correlate with previous points in the trace, as the points it did correlate with have been discarded (and replaced with non-related points). Hence, an adversary can get a ‘fingerprint’ of how correlation should look and any discrepancy of the cross-correlation will allow the adversary to distinguish real and simulated traces. For our particular attacks, we used visual inspection of the cross-correlation trace to distinguish a simulated trace from a real trace. We briefly touch on how this process could be automated in the next section.

4.3.4. Experiments on Real Devices

Side channel attacks tend to be played down in seriousness because they are both device and implementation specific. From them it is hard (or even impossible) to draw conclusions about a scheme’s side channel resistance based on a single attack. However, since the goal of Standaert et al.’s work [158] is for practitioners to be able to implement the simulator and run the experiment, we did not want to rely on pure leakage emulations. In the original paper [108] we implement, and attack, the simulator on a wide variety of devices with a wide range of implementation decisions. We show that our attack is not device specific, exploiting some particular aspect of a device or implementation but in fact is a general distinguisher showing the flaw lies with the simulator itself. To this end, we intentionally chose devices that lead to a wide variety of different AES implementations, different leakage models and different noise characteristics. Below, we
give an example of one such device to show that our distinguisher can break the simulator given by Standaert et al. Here a microcontroller utilising the 8051 instruction set is considered. The cross-correlation trace reveals detailed information about the data flow within the AES operation. Using this, it was possible to detect the order in which both the state bytes and functions were accessed. For a masked implementation it was also possible to detect when, and where, the mask was applied. Figure 4.9 shows the cross-correlation traces for both the real traces (Fig. 4.9(a)) and the simulated traces (Fig. 4.9(b)). There is a distinct difference between the two and thus it is possible for the distinguisher to differentiate between real and simulated traces, winning the \( p-q\)-sim game. We distinguished the simulated traces from the real traces via visual inspection of the cross-correlation trace. However, using cross-correlation fingerprints constructed using the Leak oracle, it would be possible to automate this process. The automated distinguisher would work by looking for discrepancies between the fingerprints and the given cross-correlation trace. A large discrepancy would indicate a simulated trace, while a cross-correlation trace similar to the fingerprint would indicate a real trace. The ‘gap’ appearing in the real correlation trace is due to the location the cut in the trace would be made. The original paper [158] states that the trace should be cut in a place of ‘low information’. We have cut between rounds where no operations are taking place. This is what can be seen in the middle of this trace (the trace is zoomed in to only the window of interest around the point of concatenation).

In our paper [108] a series of other devices, with a wide spectrum of implementation choices, are analysed to demonstrate that this attack is not just on a particular instantiation but can be
performed regardless of choice of implementation and device. Thus demonstrating that it is in fact possible to distinguish the simulator and that particular implementation choices are not being exploited.

### 4.3.5. Trying to Fix the Simulator

We give a brief overview of some of the methods we considered here, while full details can be found in the paper [108].

We considered the effects of standard DPA style countermeasures on the devices to see whether this helped the simulator. However, it was still possible to distinguish real and simulated traces. While masking may seem promising because it stops (first order) DPA attacks, it does not help protect against the distinguisher since it is only tracking correlation instead of trying to learn the underlying values. Adjusting the signal to noise ratio helped to a certain extent, since a lower signal made it harder to track correlation. However, when the noise was too high, the correlation could be tracked using the noise instead of the underlying signal.

From the point of view of adjusting the simulator, we considered several options including splitting the trace in different places and performing multiple splits in a trace. However, the split in the trace was always detectable by the cross-correlation distinguisher, for the same reasons that the split and concatenate simulator was detectable.

### 4.4. Creating a Theoretical Simulator

In the previous sections it has been shown that for real world devices it seems incredibly hard to construct a simulator. Since all devices work on data in a (somewhat) serial manner, it does not seem possible to construct any variation of the split and concatenate simulator which does not break the flow of data, allowing it to be distinguished from a real trace. We conclude by showing how to construct a simulator for a slight variation of the underlying block cipher. This variation can be plugged back into the original PRG and thus the theorem of security for the PRG will be left intact. The proof of security will now reduce to our new simulator instead of the split and concatenate simulator. We leave it as an open question as to whether such a simulator can be
simulator $S^t(x, c)$:
Perform a meet–in–the–middle attack to learn a valid $(k^*_i, k^{*'}_i)$
$\text{BC}_{k^*_i}(\text{BC}_{k^{*'}_i}(x)) \rightsquigarrow \Lambda$

Return $\Lambda$

constructed for AES (or any other commonly used block cipher).

We now discuss the variation where, instead of using a single block cipher $\text{BC}$ to create our underlying 2PRG, the double block cipher $\text{2BC}$ which is defined as $\text{2BC}_{k_i, k'_i}(x) = \text{BC}_{k_i}(\text{BC}_{k'_i}(x))$ is used. However $\text{2BC}$ cannot be plugged directly into the 2PRG construction, since now that the key (or key pair $(k_i, k'_i)$) is twice as long as the output of $\text{2BC}$, it must be called twice to update the key and as an output $x_i$ is also required, $\text{2BC}$ will be called three times instead of the twice $\text{BC}$ was required to be called. We denote the replacement for the 2PRG, 3PRG and give the tweaked construction in Fig. 4.10. From an efficiency point of view, for each output $x_i$ the blockcipher $\text{BC}$ now needs to be called six times, instead of twice as per the original scheme.

While the PRG is now more inefficient, the advantage is that the double block cipher has a natural point to split the traces (unlike the original block cipher due to information flow), in the middle, between the two block cipher calls and this will work regardless of the chosen blockcipher. The simulator is given in Fig. 4.11. Given the plaintext/ciphertext pair $(x, c)$, the simulator works by choosing two keys $(k^*_i, k^{*'}_i)$ such that $\text{2BC}_{k^*_i, k^{*'}_i}(x) = c$ using a meet-in-the-middle attack [106]. It is worth noting that we do not explicitly give the simulator a (arbitrary) pair of keys as input. To match the given framework, the simulator would be given a pair of keys which it would simply ignore. We have excluded them here for aid of clarity.
The correlation distinguisher will no longer be able to differentiate between real and simulated traces. The distinguisher works by detecting the ‘break’ in correlation where traces have been split. However, the break will not occur with this simulator; there is never a break in the state to detect. The keys are chosen in such a manner that the state between the plaintext and ciphertext is consistent the whole way through. Unlike the attempt to make the state consistent using a different round key (detailed in the full version [108]), the two keys are independent and only used once. Thus there is no change in correlation to detect here either. The disadvantage of this simulator is that it is no longer practical. A meet-in-the-middle attack for 2BC, instantiated with AES-128, takes approximately $2^{65}$ AES encryptions to find a pair of keys which meet the required criteria. This means that it is no longer an experiment which practitioners can implement and run in practice. While the simulator will be indistinguishable to the cross-correlation distinguisher given within this work, it may be possible to construct a different distinguisher to exploit certain properties. The block cipher 2BC is called 3 times in the execution of 3PRG and thus an adversary will see three leakage traces per key pair. In the real world these three traces will be on the same key pair, whereas in the simulatable world the simulator may use a different key pair per trace. This fact may be exploitable by an adversary to construct a distinguisher against our new simulator. We leave this as further research. It may be possible to prove the security of the PRG given in Fig. 4.10, in a similar manner to the original. However, due to the possible issues with the simulator, we do not give a proof of security here.

4.5. Further Work

In this section we discuss work which has appeared since the publication of our work [108] which builds upon either our work or the original work of Standaert et al. [158].

Fuller and Hamlin [71] expand on the simulatable leakage model, as discussed in more detail in Ch. 3. An open question is if it is easier to construct simulators in one of the models given here, since they are more flexible to choose the conditions of when things leak and the power of the adversary. However, since they do not propose any new constructions for simulators within the work, it is not discussed in more detail here.
4.5.1. Alternate Simulators

Pessl et al. [132] argue that the cross-correlation distinguisher exploited certain properties and proposed an efficient simulator which is indistinguishable against our correlation based distinguisher. The properties that they believe are exploited by the distinguisher are recalled here. We then argue why not all of the assumptions about the distinguisher are completely accurate and how this could possibly lead to a break in their proposed simulator.

They start with an investigation of where the intra-trace correlation arises from. It is stated that the AES algorithm itself is unlikely to be the cause of the correlation observed by our distinguisher since the correlation between the input and output of the SubBytes step is suitably small. This assumption has several issues; firstly, while it is true that the correlation between the input and the output of the SBox (by algorithmic design) is low, this is not the case for other points in the algorithm, such as the input to ShiftRows and the input to MixColumns (which are the same, so are trivially highly correlated). This may mean that it is possible to split a trace between the input and output of the SBox but we have shown that in practice this is in fact hard to do [108]. Secondly, the assumption is that low correlation (\(<0.5\)) versus high correlation is being exploited by the distinguisher. This is not strictly true as the distinguisher is actually using comparatively high correlation versus low correlation. For example, if correlation is 0.1 this can be seen as ‘low’ but if the other correlation scores are 0.04 then it is in fact high. Therefore, our distinguisher is able to work in scenarios when the correlation is never greater than 0.5.

Pessl et al. [132] then construct a new simulator for AES. The simulator is given in Fig. 4.12, where the triangles \(\blacktriangleleft, \blacktriangleright\) are used to represent a weighted sum (decreasing and increasing respectively). In the resulting trace, the first point will be the first point from the trace encrypting \(x\), while the last point in the simulated trace will be the last point from the trace that encrypts to \(c\).
Formally, if a trace has \( n \) points then the \( i \)th point is calculated as
\[
I_a[i] = (1 - \frac{i}{n}) \cdot I_a[i] + \frac{i}{n} \cdot I_b[i].
\]
The intuition is that since the trace is never split, there will not be a ‘dip’ in correlation for the cross-correlation distinguisher to detect, therefore providing a secure simulator. However, we believe a simulated trace from this simulator will still be distinguishable from a real trace. Firstly, we believe the addition of two traces will weaken the correlation across the entire trace, compared to a real trace, and thus it would be detectable by the cross-correlation distinguisher. However, even if it is indistinguishable to the cross-correlation distinguisher, we believe there are other ways to distinguish between a real and simulated trace. Since the simulated trace is the addition of two traces, the mean will be increased over the original trace and the variance will be affected as well. Therefore, it would be possible to construct a distinguisher using one of these metrics. Hence, we believe this will not be a simulator for AES power traces.

4.5.2. Schemes Built Upon Simulators

Pereira et al. [131] assume the existence of a simulator for a PRF and then ask what schemes can be built on top of it which are provably secure in the face of leakage. They produce a leakage resilient MAC and a leakage resilient encryption scheme on top of a leakage resilient PRF (which is secure in the simulatable leakage sense).

The PRF is run in an iterative fashion; message block \( M_i \) is put through the PRF using key \( k_i \) to produce key \( k_{i+1} \). If the message is \( n \) blocks, the tag is simply \( k_{n+1} \). To allow the scheme to be run multiple times and still meet the \( q \)-simulatability requirements, a leak free component is needed. Otherwise, by calling the MAC \( q + 1 \) times, the key \( k \) could be leaked on \( q + 1 \) times, breaking the \( q \)-sim requirement. This component is a PRF which takes in the master key and an IV and outputs \( k_0 \) to be used for that round. Since this cannot leak, the master key can never have any leakage output on it and all other keys \( k_i \) will meet the \( q \)-sim requirement since the keys get updated after a single use.

The question arises; if there is a leak free PRF why try to construct anything else instead of using this for encryption or for the MAC? To construct something that does not leak will be extremely expensive and could be orders of magnitude slower to run. Therefore, the goal is to minimise the number of calls to this component.
The encryption scheme uses the PRG proved within the original paper by Standaert et al. [158] but instead of outputting the pseudorandom stream, exclusive-or it with the message to produce the ciphertext. To perform multiple encryptions, the leak free component is used to take in the master key and an IV and output a ‘session key’ to be used for the encryption.

While their work extends the schemes which can be created using a leakage simulator for a PRF, it relies on some fairly strong assumptions such as a leak free component which does not model practice. An open question is if it is possible to remove the need for the leak free component or any of the other strong assumptions.

4.6. Conclusions

We show that the simulator given by Standaert et al. [158] can in fact be distinguished from real leakage before showing how to adjust the underlying scheme such that a theoretical simulator can be constructed. It remains an open question as to how to construct a practical simulator such that practitioners can implement the simulator and play the $p$-$q$-sim game using real devices. However, based on the work given in this chapter, we feel that this may in fact be an unobtainable goal unless new techniques arise.

The work of Dinur et al. [53] on meet in the middle attacks and making them more efficient for different numbers of encryption calls can be applied directly to our simulator. This makes our theoretical simulator proportionally more efficient compared to brute force but does not bring the simulator any closer to being practical.

Recently, work has started to use the simulatable leakage model to create schemes which rely on the existence of a simulator [131]. While the existence of our theoretical simulator means the schemes are secure, it does not meet the original goal, of the model, of being practically testable by the evaluation labs. The more schemes that are designed, the more important it becomes to find such a simulator which meets these requirements, so that the leakage game can be played in evaluation labs to encourage the use of schemes which are secure in the simulatable leakage model.
5. Continuous Leakage Model

In the previous chapter, we demonstrated the difficulty of constructing an efficient simulator for AES-128. Without a simulator, any proofs within the simulatable leakage model do not provide any security guarantees. Hence, we must consider constructing schemes within a different leakage model. The model we have chosen to investigate is the continuous leakage model using the OCLI assumption. We have chosen this model as the continuous leakage corresponds with how a device leaks in practice and the adaptive leakage functions corresponds to an adversary’s ability to take different measurements on different invocations.

In this chapter we start by formalising the leakage resilient model for the case when each function (that is required to be leakage resilient) is split into two components and each component leaks individually (using the OCLI assumption). Therefore, two leakage functions will be provided each time a function is required to be evaluated. We construct two leakage resilient MACs, and a leakage resilient PRF. The proofs of security for these schemes use extremely similar techniques to each other. Therefore, our proof technique may be of interest to show the security of other schemes in the face of leakage. We show how to construct various leakage resilient (authenticated) encryption schemes from these primitives. We implemented one of the schemes in order to evaluate its performance and investigate whether the resulting power traces conform to the assumptions made within the proof. We extrapolate the lessons learnt from implementing this scheme to the other schemes. To our knowledge we are the first to investigate the power traces of a theoretical leakage resilient scheme.

109
5.1. The Leakage Resilient Security Model

In this section we describe on a more formal level the leakage model and the set of leakage functions, to be used throughout the remainder of this chapter. Security definitions involving this leakage will be given as required, later in the chapter.

Micali and Reyzin introduced the Only Computation Leaks Information (OCLI) assumption [120]. The assumption states that only data currently being computed on leaks, while any data that is at rest (e.g. just being stored in memory) will not leak (see Sect. 3.6.2 for more details). There are particular scenarios in which the assumption has been shown not to hold, such as for certain technologies at gate level [138]. However, it sufficiently captures the behaviour of many of the current state of the art devices and appears to hold for the scenarios we will be considering. A leak-free component is extremely hard/impossible to implement on the majority of devices and, therefore, the OCLI assumption in preferable.

To utilise the OCLI assumption, the computation will be split into smaller computations such that each computation only uses a portion of the secret data available, thus restricting the scope of what can be leaked upon. OCLI will be modelled in this chapter by splitting a function $F$ (that needs to be made leakage resilient) into two separate functions denoted $F^\downarrow$ and $F^\uparrow$. The part of the secret information $S$ accessed by the function $F^\downarrow$ will be denoted $S^\downarrow$, while the part of the secret accessed by $F^\uparrow$ will be denoted $S^\uparrow$.

Concretely, the adversary may, per function call, adaptively choose leakage functions $l^\downarrow$ and $l^\uparrow$ which will leak up to $\lambda$ bits each (where $\lambda$ will be a security parameter) on $F^\downarrow$ and $F^\uparrow$ respectively. The adversary then receives the output $l^\downarrow(S^\downarrow, x^\downarrow, R^\downarrow)$ and $l^\uparrow(S^\uparrow, x^\uparrow, R^\uparrow)$, where $x^\downarrow$ and $x^\uparrow$ are the inputs to the functions, and $R^\downarrow$ and $R^\uparrow$ are the internal randomness used within the functions ($F^\downarrow$ and $F^\uparrow$). This formalism, of how a function is split and leaked upon, can be visualised in Fig. 5.1

If the leakage process is iterated multiple times, subscripts will be used to denote the iteration that is currently being operated on. For example, $S_1^\downarrow, S_1^\uparrow, R_1^\downarrow, R_1^\uparrow, l_1^\downarrow, l_1^\uparrow$. The leakage functions $l_i^\downarrow$ and $l_i^\uparrow$ may depend on the outcome of leakage functions $l_j^\downarrow, l_j^\uparrow$ for $j < i$ (the leakage functions may depend on leakage from a previous call). This corresponds to the adversary adjusting the measurement set up after the complete execution of the device.
The leakage functions $l_i^G$ and $l_i^G$ can be chosen adaptively from round to round and can depend on the leakage seen in previous rounds. However, the two leakage functions must be chosen at the same time for a single round. This mild restriction, that $l_i^G$ cannot depend on the leakage from $l_i^G$, is fairly common in the literature [75, 101] and reflects the capabilities of a real world adversary (the adversary cannot set up, or adjust, measuring equipment halfway through the execution of a device).

At one particular point in this chapter we will require three shares instead of two (the reason will be justified at the time); this third function, denoted $F^\circ$, will work with secret $S^\circ$ and randomness $R^\circ$, and the adversary will supply leakage function $l^\circ$.

Given the above restrictions on the leakage, we define the valid set of leakage functions for $F$ (when split into $(F^\circ, F^\circ)$) as the set $\mathcal{L}_F$, where:

$$\mathcal{L}_F = \{l | l : S \times X \times R \rightarrow \{0, 1\}^{2\lambda}, l((S^\circ, S^\circ), x^\circ, (R^\circ, R^\circ)) = l^\circ((S^\circ, x^\circ, R^\circ)) \lor (S^\circ, x^\circ, R^\circ)\}$$
where $S, \mathcal{X}, \mathcal{R}$ are the share space, input space and randomness space respectively. The functions $l^\mathcal{G}_i$ and $l^\mathcal{R}_i$ are restricted to only being able to output $\lambda$ bits each. By modelling the restrictions discussed above formally, it enables security notions to restrict the type of leakage, to come from a certain set, in a manner which is clear in the security definition.

5.1.1. Leakage and the Generic Group Model

The majority of proofs within the leakage resilient model, and the proofs within this work, are within the GGM. The GGM was not designed to consider leakage and therefore we must discuss how leakage can be incorporated. To achieve this, it must be decided how the representations are chosen such that leakage relates to practice.

Leaking on generic group elements only reveals information about the group element’s representation. In some proofs (without leakage) that use the GGM, the representation of group elements can be chosen in such a way that even sampling a random group element is hard (for an adversary). This is typically achieved by representing group elements as ‘long’ random strings. When leakage is included in proofs, such a strategy would not make sense because it would imply that only ‘large’ amounts of leakage would strengthen the adversary. For example, it may mean that even given $\lambda$ bits of leakage such that $\lambda$ is considerably greater than the number of bits required to represent the group ($\log p$), the adversary can still not learn the group element. This clearly does not represent how a device behaves in practice and from a theoretical point of view the size of the generic group representation should not affect the security of a scheme. We therefore instantiate the generic group model using compact representations instead. By setting $\Xi_i = \{0, 1\}^n$ where $n = \lceil \log p \rceil$, the group elements will all have the unique representations required. This gives the adversary the ability to sample group elements efficiently and directly, and with these compact representations a bit of leakage on the representation corresponds to a bit of leakage on the underlying element. This helps the scheme behave in a way that closer aligns with practice.

In contrast, Kiltz and Pietrzak [101] (and similarly, Galindo and Vivek [75]) use indirect sampling by raising some generator to a random exponent. They allow leakage on both the random representations, as well as the points’ discrete logarithms (relative some generator), to
model the adversary’s ability to leak on the sampling computation itself. The proofs given in this thesis can be seen as more restrictive, as they only hold for implementing the sampling directly. However, this is not a problem in practice, because it is possible to sample random elliptic curve points efficiently without performing an exponentiation with an unknown exponent. Methods for sampling group elements are discussed in more detail in Sect. 5.6.

**Practical Restrictions of the GGM**

While the GGM is required to theoretically prove our schemes secure, in the face of leakage, it is important to be aware where the model deviates from practice. The GGM is designed so that the adversary is unable to exploit the structure of the underlying group. A consequence of this is that the adversary will not be able to exploit any leakage from the computation of the group elements, only leakage on the input and/or output to the generic group oracles. This can be seen as implicitly assuming that the group oracles act as leak free components. However, in practice it may be possible for an adversary to exploit leakage from these computations. Therefore, the group operations must either be implemented in a leak free manner, be suitably protected utilising side channel countermeasures or implemented and tested to verify that the leakage from these operations is not exploitable.

However, we feel that having the group operations being leak free is practically more achievable than having larger components be leak free, such as the key update mechanism.

The generation of a random group element is also implicitly assumed to occur in a leakage free manner within this model. In Sect. 5.6.3 we investigate different methods of generating random elements within an elliptic curve group such as to minimise the exploitable leakage, thus bringing the randomness generation closer to the assumption. After discussing several candidate methods, we choose one and check that we can not detect any leakage within the power trace.

**5.2. Previous Work in the Continuous Leakage Model**

In the continuous leakage model there has been a series of work which all use a similar technique to prove the leakage resilience of the schemes [74, 75, 101, 115, 159]. These schemes split the
Figure 5.2.: KP KEM K\textsuperscript{[101]}
Figure 5.3.: KP Leakage Resilient KEM $K^*$ [101]

output from $\text{Dec}^{\otimes}(\cdot)$ can be seen below:

$$\begin{align*}
  k_i &= k_i^\otimes \cdot k_i^\otimes \\
  &= e(S_i^\otimes, C) \cdot e(S_i^\otimes, C) \\
  &= e(S_i^\otimes \cdot S_i^\otimes, C) \\
  &= e(X, C) \\
  &= \text{Dec}(sk, C)
\end{align*}$$

Thus, providing that the two shares always multiply together to give the key, the leakage resilient version of the scheme computes the same as the non-leakage resilient version. Below, it is shown that the key is updated in such a way that the two shares $S_i^\otimes, S_i^\otimes$ always multiply together to return the original secret key.
\[ S_i^\bullet \cdot S_i^\bullet = S_0^\bullet \cdot \prod_{j=1}^{i} R_j^\# \cdot S_0^\bullet \cdot \prod_{j=1}^{i} (R_j^\#)^{-1} \]

\[ = S_0^\bullet \cdot \prod_{j=1}^{i} R_j^\# \cdot S_0^\bullet \cdot \prod_{j=1}^{i} (R_j^\#)^{-1} \]

\[ = S_0^\bullet \cdot S_0^\bullet \cdot \prod_{j=1}^{i} R_j^\# \cdot \prod_{j=1}^{i} (R_j^\#)^{-1} \]

\[ = S_0^\bullet \cdot S_0^\bullet \]

\[ = S_0^\bullet \cdot (S_0^\bullet)^{-1} \cdot X \]

\[ = X \]

\[ = sk \]

Hence the shares always recombine to give the initial key, showing that both the leakage resilient and original variants of the scheme compute the same results.

The cost of making the entire scheme leakage resilient is fairly low. The key generation algorithm must generate a second random group element and perform an inversion and multiplication. The encryption function stays the same, while the decryption function is essentially performed twice, adding an extra pairing, an extra random element generation, a group inverse and three group multiplications.

The leakage resilience of this scheme (CCLA1 security [101]) is shown within the GGM [101]. For completeness, as well as future reference, the theorem statement is given below. The theorem statement is given using the set notation and the given set from Sect. 5.1. \( L_{\text{dec}} \) represents that leakage will be defined on input to the pair of functions \((\text{Dec}^\bullet, \text{Dec}^\#)\).

**Theorem 2** ([101, Lemma A.3]). The KEM \( K^* \) is \( L_{\text{dec}} \)-CCLA1 secure in the GGM. The advantage of a \( q \)-query (to the generic group oracles) adversary \( A \) who is allowed \( \lambda \) bits of leakage (to both \( \text{Dec}^\bullet \) and \( \text{Dec}^\# \)) is given by:

\[ \text{Adv}_{K^*}^{L_{\text{dec}} \text{-ccla1}}(A) \leq \frac{2^{2\lambda+1} \cdot q^3}{p} \]
\textbf{proc KG:}\n\begin{align*}
x, x_0, x_1 & \leftarrow Z_p \\
x & \leftarrow g_1^x \\
X_0 & \leftarrow g_1^{x_0} \\
X_1 & \leftarrow g_1^{x_1} \\
X_T & \leftarrow e(g_1, g_1)^x \\
\end{align*}
\begin{align*}
& pk \leftarrow (X_T, X_0, X_1) \\
& sk \leftarrow (X, X_0, X_1) \\
& \text{Return } (pk, sk)
\end{align*}

\textbf{proc Sign}(sk, M):
\begin{align*}
(X, X_0, X_1) & \leftarrow sk \\
t & \leftarrow Z_p \\
\sigma & \leftarrow (X \cdot (X_0 \cdot X_1^M)^t, g_1^t) \\
& \text{Return } \sigma
\end{align*}

\textbf{proc VRFY}(pk, M, \sigma):
\begin{align*}
(X_T, X_0, X_1) & \leftarrow pk \\
(\sigma_1, \sigma_2) & \leftarrow \sigma \\
Y & \leftarrow X_T \cdot e(\sigma_2, X_0 \cdot X_1^M) \\
Z & \leftarrow e(\sigma_1, g_1) \\
& \text{Return } (Y = Z)
\end{align*}

Figure 5.4.: GV Signature scheme S [75]

Theorem 2 implies that bounding an adversary’s advantage to at most \(2^{-n}\) requires \(\lambda \leq \frac{1}{2}(\log p - 3 \cdot \log q - n)\). Using a prime of 160 bits and requiring 80 bits of security results in \(\lambda = 40 - \frac{3 \log q}{2}\), implying that when the adversary is allowed \(q = 2^{20}\) queries they can learn 10 bits, or approximately a sixteenth of an element, per leakage function \(l\), giving 20 bits of leakage total.

\subsection*{5.2.2. Galindo and Vivek Signature Scheme [75]}

Inspired by the work of Kiltz and Pietrzak [101], Galindo and Vivek [75] create a signature scheme \(S\), which is given in Fig. 5.4. The scheme is shown to be Existential Unforgeability secure [75, Theorem 1] (See [99] for signature security definitions). This uses the same method as Kiltz and Pietrzak to update and share the key. Sign is the only algorithm which uses the secret key (VRFY uses the public key) and, therefore, only Sign will be split into two parts.

Since the signature scheme does not require a pairing to sign the message, it is possible to recombine the two halves of the computation to calculate the same signature as the non-leakage resilient scheme using the properties of the sharing, thus making the scheme leakage resilient.

The leakage resilient variation \(S^*\) is given in Fig. 5.5.

As with the KEM, a symmetric pairing is used for this scheme. Since the message is used as an exponent it must be in \(Z_p\). However, it is possible to sign arbitrary messages by using a (collision-resistant) hash function which hashes from bit strings to \(Z_p\). Correctness of the leakage resilient scheme follows from the fact that Sign computes the same as for the non-
proc KG*:
\[
x, x_0, x_1 \leftarrow Z_p^\times
\]
\[
X \leftarrow g_1^x
\]
\[
X_0 \leftarrow g_1^{x_0}
\]
\[
X_1 \leftarrow g_1^{x_1}
\]
\[
X_T \leftarrow e(g_1, g_1)^x
\]
\[
S_0 \leftarrow X \cdot (S_0^-)^{-1}
\]
\[
X_T \leftarrow e(g_1, g_1)^x
\]
\[
S_0 \leftarrow X \cdot (S_0^-)^{-1}
\]
\[
pk \leftarrow (X_T, X_0, X_1)
\]
\[
sk \leftarrow (S_0^\times, S_0^\times, X_0, X_1)
\]
Return \((pk, sk)\)

proc Sign\(^{-}\)((S_i\^\times, X_0, X_1), M)\):
\[
t \leftarrow Z_p^\times
\]
\[
\sigma_1 \leftarrow g_1^t
\]
\[
\sigma_i \leftarrow S_i^\times \cdot (X_0 \cdot X_1^M)^t
\]
\[
R_i^{+1} \leftarrow G_1
\]
\[
S_i^{+1} \leftarrow S_i^\times \cdot R_i^{+1}
\]
Return \((S_i^{+1}, \sigma_i, \sigma_1, R_i^{+1})\)

proc VRFY\((pk, M, \sigma)\):
\[
(X_T, X_0, X_1) \leftarrow pk
\]
\[
(\sigma_1, \sigma_2) \leftarrow \sigma
\]
\[
Y \leftarrow X_T \cdot e(\sigma_2, X_0 \cdot X_1^M)
\]
\[
Z \leftarrow e(\sigma_1, g_1)
\]
Return \((Y = Z)\)

Figure 5.5.: GV Leakage Resilient Signature scheme \(S^*\) [75]

leakage resilient scheme (VRFY remains unchanged since all the information used is public),
which can be seen as follows:

\[
\sigma_i = (\sigma_1, \sigma_2)
\]
\[
= (S_i^\times \cdot \sigma_i^\times, g_1^t)
\]
\[
= (S_i^\times \cdot S_i^\times \cdot (X_0 \cdot X_1^M)^t, g_1^t)
\]
\[
= (X \cdot (X_0 \cdot X_1^M)^t, g_1^t)
\]
\[
= \text{Sign}(sk, M)
\]

Hence, using the fact above that the shares always recombine to give the original secret key, correctness of the scheme follows. As with the KEM, the key generation algorithm must generate a second random group element and perform an inversion and multiplication. The sign algorithm must generate an extra random group element, and perform an extra inversion and three group multiplications. Since all information in VRFY is public, the algorithm remains unchanged. Thus the cost of making the scheme leakage resilient is fairly low.
The leakage resilience of this scheme (EUL security [75]) is shown within the GGM [75]. For completeness, as well as future reference, the theorem statement is given below.

**Theorem 3** ([75, Theorem 2]). The signature scheme $S^*$ is Existential Unforgeability with Leakage ($L_{\text{sign}}$-EUL) secure in the GGM. The advantage of a $q$-query (to the generic group oracles) adversary $A$ who is allowed $\lambda$ bits of leakage (to both $\text{Sign}^\ast$ and $\text{Sign}^\ast$) is given by:

$$\text{Adv}_{S^*_{\text{sign}}-\text{eul}}(A) \leq O\left(\frac{2^{2\lambda} \cdot q^2}{p}\right)$$

Theorem 3 implies that bounding an adversary’s advantage to at most $2^{-n}$ requires $\lambda \leq \frac{1}{2}(\log p - 2 \cdot \log q - n)$. Despite the different purposes of this scheme, compared to the KEM, due to their similarities, the bounds and therefore the leakage allowed are extremely similar.

### 5.2.3. Other Signature Schemes

There have since been more signature schemes produced within the leakage resilient model [74, 159]. However, due to their similarity to the scheme given above and their use of the techniques first used within the Kiltz and Pietrzak paper, they will not be discussed in further detail here.

### 5.3. A Key Update Mechanism

In the previous section it can be seen that all of the given work relies on the same method of updating the key. In this section we formulate the notion of a key update mechanism and show that the method of updating keys proposed by Kiltz and Pietrzak [101] meets this definition. We then provide two security notions for a key update mechanism, a computational notion and a decisional notion. We explain why these notions are desirable and what they capture. We conclude this section by showing that the Kiltz-Pietrzak (KP) key update mechanism meets the computational notion of security but not the decisional notion. Before this work, there were no notions for securely updating a key in the face of leakage. This made updating the key extremely hard to reason about and thus it was only considered within the context of a particular scheme. With these notions in place it is now possible to reason about the key update mechanisms in a
standalone manner. Reasoning about the key update mechanism gives a starting point to show
the security of schemes built upon them.

**Definition 30 (Key Update Mechanism).** A key update mechanism is defined as the tuple $\mathcal{KU} = (\text{Share}, \text{Recombine}, \mathcal{U}^\leftarrow, \mathcal{U}^\rightarrow)$ such that; Share is a random algorithm which takes in a key $k$ and returns two shares $(S_0^\leftarrow, S_0^\rightarrow)$, this is denoted $(S_0^\leftarrow, S_0^\rightarrow) \overset{\$}{\leftarrow} \text{Share}(k)$. The two update algorithms $\mathcal{U}^\leftarrow, \mathcal{U}^\rightarrow$ are always called as a pair. $\mathcal{U}^\leftarrow$ takes in a share $S_i^\leftarrow$ and returns an updated share $S_{i+1}^\leftarrow$ and some additional information $O_i^\leftarrow$. $\mathcal{U}^\rightarrow$ takes in a share $S_i^\rightarrow$ and the additional information $O_i^\rightarrow$ and returns an updated share $S_{i+1}^\rightarrow$. These are denoted $(S_{i+1}^\leftarrow, O_i^\leftarrow) \overset{\$}{\leftarrow} \mathcal{U}^\leftarrow(S_i^\leftarrow)$ and $S_{i+1}^\rightarrow \overset{\$}{\leftarrow} \mathcal{U}^\rightarrow(S_i^\rightarrow, O_i^\rightarrow)$ respectively. Finally, Recombine takes in two shares $S_i^\leftarrow, S_i^\rightarrow$ and returns a key $k_i$, this is denoted $k_i \leftarrow \text{Recombine}(S_i^\leftarrow, S_i^\rightarrow)$. For correctness it is required that $\text{Recombine}(\text{Share}(k)) = k$. Recombine is a relic of the proof and is never actually called in the schemes. It will be used to argue correctness. The algorithms are visualised in Fig. 5.6.

We define an equivalence class as follows; we say $(S_i^\leftarrow, S_i^\rightarrow) \equiv (S_j^\leftarrow, S_j^\rightarrow)$ if $\text{Recombine}(S_i^\leftarrow, S_i^\rightarrow) = \text{Recombine}(S_j^\leftarrow, S_j^\rightarrow)$. Then the final requirement for $\mathcal{KU}$ to be a key update mechanism is that the algorithms $\mathcal{U}^\leftarrow$ and $\mathcal{U}^\rightarrow$ preserve the equivalence class of the shares (and thus $\forall i : k_i = k$). Formally we require $(S_i^\leftarrow, S_i^\rightarrow) \equiv (S_{i+1}^\leftarrow, S_{i+1}^\rightarrow)$ where $(S_{i+1}^\leftarrow, O_i^\leftarrow) \overset{\$}{\leftarrow} \mathcal{U}^\leftarrow(S_i^\leftarrow), S_{i+1}^\rightarrow \overset{\$}{\leftarrow} \mathcal{U}^\rightarrow(S_{i+1}^\leftarrow, O_i^\rightarrow)$. 

Figure 5.7 shows how the method of sharing and updating a key designed by Kiltz and
Pietrzak [101] can be represented as a key update mechanism, which will be denoted $\mathcal{KU}$. The Share function generates two random shares such that they multiply together to give back the key. The update function $U^\circ$ takes in a share $S^\circ_i$ and multiplies it by a random group element $R_{i+1}^\circ$ to generate the new share $S_{i+1}^\circ$. The second update function $U^\bullet$ takes in a share $S^\bullet_i$ and the randomness $R_{i+1}^\bullet$ (this is the $O_i^\circ$ from the definition) output by the previous update function and multiplies the share by the inverse of the randomness to generate the new share $S_{i+1}^\bullet$. The Recombine function simply takes in two shares and multiplies them together to get the key. Below, we show that $\mathcal{KU}$ meets the definition of being a key update mechanism.

**Claim 1.** $\mathcal{KU}$ defined above is a key update mechanism, over a group $\mathbb{G}_1$.  

**Proof.** It has to be shown that for any shares $(S^\circ_i, S^\bullet_i)$ that are output from the functions $(U^\circ, U^\bullet)$ (after iterating multiple times if required), on original shares $(S^\circ_0, S^\bullet_0)$ from Share($k$), recombine to return $k$. That is, it needs to be shown that for all output of the update functions, after calling the update functions a number of times, Recombine($S^\circ_i, S^\bullet_i$) = $k$. 

**Figure 5.7.: KP Key Update Mechanism $\mathcal{KU}$**
Recombine\((S_i^\bullet, S_i^\bullet)\)  
\[ = S_i^\bullet \cdot \prod_{j=1}^{i} R_j^\bullet \cdot (\prod_{j=1}^{i} (R_j^\bullet)^{-1}) \] 
\[ = S_0^\bullet \cdot \prod_{j=1}^{i} R_j^\bullet \cdot (\prod_{j=1}^{i} R_j^\bullet)^{-1} \] 
\[ = S_0^\bullet \cdot \prod_{j=1}^{i} R_j^\bullet \cdot (\prod_{j=1}^{i} (R_j^\bullet)^{-1}) \] 
\[ = S_0^\bullet \cdot S_0^\bullet \cdot (\prod_{j=1}^{i} R_j^\bullet)^{-1} \] 
\[ = S_0^\bullet \cdot k \cdot (S_0^\bullet)^{-1} \] 
\[ = k \]

Hence the update functions preserves the equivalence and \(\text{Recombine}\) recovers the key originally shared. Thus, \(\mathcal{KU}\) is a key update mechanism as required.

Since a key update mechanism will not be used in a standalone manner, but will be used within a greater system, we now describe what it means for a function to be compatible with a key update mechanism. We denote this notion as \(\text{key update splittable}\).

**Definition 31** (Key Update Splittable). A tuple of functions \((F^\bullet, F^\bullet)\) is denoted a split of \(F\) conforming to key update mechanism \(\mathcal{KU}\) if the following two properties hold. Firstly:

\[ \{ F(k, x) \}_R = \{ F^*(\text{Share}(k), x) \}_{R^*} \]

where \(F^*\) is defined in Fig. 5.8. The equivalence is over the randomness from sets \(R, R^*\) used by \(F, F^*\) respectively. Secondly, that for all sharings \((S_0^\bullet, S_0^\bullet)\) the joint distribution on \((S_1^\bullet, S_1^\bullet)\), after \(F^*\) has been called once, is the same as if \((S_0^\bullet, S_0^\bullet)\) had been updated using \((U^\bullet, U^\bullet)\).

If there exists such a tuple \((F^\bullet, F^\bullet)\) then \(F\) is key update splittable.

All functions which are shared using the key update mechanism, will be called with the same input/output behaviour, as in Fig. 5.8 unless otherwise stated. In all the previous work given,
\[
\text{proc } F^*(S^i, S^{i+1}, x):
\]
\[
(S^i_{t+1}, O^i_{t+1}) \leftarrow F^*(S^i, x)
\]
\[
(S^i_{t+1}, y) \leftarrow F^*(S^i, O^i_{t+1})
\]

Return \( y \)

Figure 5.8.: The algorithm \( F^* \)

when a function is shared, it is also called in this manner

Now that the definitions are in place, we give two security notions for a key update mechanism. Let \( \mathcal{L} \) be the set of leakage functions security is to be shown for. The first notion is Computational Key Recovery Under Leakage (\( \mathcal{L} \)-CKR-UL) and considers if an adversary can recover the key using only leakage (with leakage functions from \( \mathcal{L} \)) from the two update functions \( U^i, U^{i+1} \). It is important that a key update mechanism meets this security definition otherwise any scheme built on top of the key update mechanism will not be secure. An adversary can use the leakage on the update functions to learn the key, at which point the security games tend to be trivial to win (since Kerckhoff's principle says a crypto-system should be secure if everything bar the key is known, this implies learning the key will break the security of the scheme). Therefore, the hardness of \( \mathcal{L} \)-CKR-UL is necessary for the security of a crypto-system, with leakage, using a key update mechanism. The second notion is Decisional Key Recovery Under Leakage (\( \mathcal{L} \)-DKR-UL) in which the adversary is given one of two keys, and using the leakage from the two update functions \( U^i, U^{i+1} \), the adversary must figure out if the key they were given is the one being updated and leaked upon.

The advantage of the \( \mathcal{L} \)-DKR-UL style notion is that because the adversary cannot tell the difference between leakage on a real key and leakage on a fake key when they have been given the key, they would not be able to tell the difference between leakage on the real key and the fake key when they are only given plaintext/ciphertext pairs (for example). Hence, using this style of technique, it may be possible to reduce the security when leakage is involved, to the security when leakage is not involved plus the \( \mathcal{L} \)-DKR-UL security of the underlying key update mechanism. The result of this would be that, providing the leakage of the entire function could be simulated by the key update leakage (so that the reduction can pass the leakage function for
the scheme through to the leakage oracle of the key update function), any function which was compatible with a ℒ-DKR-UL secure key update mechanism would inherit the leakage resilience without having to reprove the scheme secure. Therefore, the hardness of ℒ-DKR-UL, for a key update mechanism, is sufficient to show the security of any scheme under leakage, which uses the key update mechanism. Unfortunately the KP key update mechanism used in this work does not satisfy the ℒ-DKR-UL notion, for the leakage set considered. Since, we cannot construct a key update mechanism which meets this security notion, we do not consider generic proofs utilising the notion for the remainder of this work. We leave it as an open question to find a key update mechanism which meets this requirement.

**experiment** \( \text{Exp}^{\text{L-ckrul}}_{\mathcal{KU}}(A) \):

\[
\begin{align*}
&k \leftarrow \mathcal{KG} \\
&(S_0 \leftarrow S_0, S_0) \leftarrow \text{Share}(k) \\
&k' \leftarrow A \text{Update}(\cdot)() \\
&\text{Return } (k = k')
\end{align*}
\]

**proc** Update(\( l_i \)):

\[
\begin{align*}
&\text{if } l_i \notin \mathcal{L} \text{ then} \\
&\text{Return } \\
&\text{end if} \\
&(l_i \leftarrow l_i) \\
&(S_{i+1} \leftarrow U_{i+1}(S_i)) \\
&S_{i+1} \leftarrow U_{i+1}(S_i, O_i) \\
&\Lambda_{i+1} \leftarrow l_i(S_i, O_i) \\
&\Lambda_{i+1} \leftarrow l_i(S_i, O_i, R_{i+1}) \\
&\text{Return } (\Lambda_i, \Lambda_{i+1})
\end{align*}
\]

Figure 5.9.: ℒ-CKR-UL experiment

**Definition 32** (Computational Key Recovery Under Leakage (ℒ-CKR-UL)). Let \( \mathcal{KU} = (\text{Share}, U^\circ, U^\circ, \text{Recombine}) \) be a key update mechanism. Then Fig. 5.9 defines the ℒ-CKR-UL security game. The advantage of an adversary \( A \) winning the game is defined as \( \text{Adv}^{\text{L-ckrul}}_{\mathcal{KU}}(A) = \Pr[\text{Exp}^{\text{L-ckrul}}_{\mathcal{KU}}(A) = 1] \).

The ℒ-CKR-UL notion given above uses the two leakage functions, one for each of the two update functions \( U^\circ, U^\circ \). Since the key update mechanism is explicitly designed to capture the OCLI assumption, it makes sense to give the security notions using the two leakage functions. If required, the security game can be adjusted to accommodate more general classes of leakage.

124
The security games given in the remaining sections of this chapter will not be restricted to the set of leakage functions which contains pairs of bounded leakage functions. The schemes we give utilise the key update mechanism and therefore security will be shown against pairs of bounded leakage functions. However, it is certainly possible to design schemes without this restricted leakage set. Thus, it is desirable to give a general notion wherever possible. Since the DKR-UL notion also aims to capture the OCLI assumption, it will be given below with the two leakage functions, one for each of the two update functions $U^\oplus, U^\ominus$.

**Definition 33** (Decisional Key Recovery Under Leakage ($L$-DKR-UL)). Let $\mathcal{KU} = (\text{Share}, U^\oplus, U^\ominus, \text{Recombine})$ be a key update mechanism. Then Fig. 5.10 defines the $L$-DKR-UL security game. The advantage of an adversary $A$ winning the game is defined as $\text{Adv}_{L-\text{DKR-UL}}(A) = \Pr[\text{Exp}_{L-\text{dkrul}}^\mathcal{KU}(A, 1) = 1] - \Pr[\text{Exp}_{L-\text{dkrul}}^\mathcal{KU}(A, 0) = 1]$.

In the $L$-DKR-UL notion, $k_0$ is always shared and leaked upon, while the adversary is given one of two keys and must figure out if the leakage corresponds to the given key. An alternative description could be given in which the adversary is always given $k_0$ and one of the two key is randomly shared and leaked upon. These two notions are equivalent, although one description may be preferable to another in certain situations.

This section is concluded with a discussion of which of these security definitions the KP key update mechanism $\mathcal{KU}$ meets. The KP key update mechanism must meet the $L_{\text{update-CKR}}$-
UL notion otherwise it would not have been possible to prove any schemes secure which use \( \mathcal{KU} \). We confirm this intuition below. The more interesting question is if it meets \( \mathcal{L} \)-update\(^{-}\)DKR-UL, due to the advantages listed above. However, we show that this is not the case by providing an adversary against \( \mathcal{KU} \) who can win the \( \mathcal{L} \)-update\(^{-}\)DKR-UL game with extremely high probability. We leave it as an open question if the security of a scheme can be related to its security without leakage and the \( \mathcal{L} \)-update\(^{-}\)CKR-UL security of \( \mathcal{KU} \). However, the attack against the Pseudo Random Leaky Function (PRLF) which uses \( \mathcal{KU} \) (see Sect. 5.5.1), makes us believe that such generalised statements are not possible for the KP key update mechanism.

**Theorem 4.** The key update mechanism \( \mathcal{KU} \) is \( \mathcal{L} \)-update\(^{-}\)-CKR-UL secure in the GGM. The advantage of a \( q \)-query (to the oracles) adversary \( A \) who is allowed \( \lambda \) bits of leakage (to both \( U^{\bullet} \) and \( U^{\circlearrowright} \)) is given by:

\[
\text{Adv}_{\mathcal{KU}}^{\mathcal{L}
-\text{update}-\text{ckrul}}(A) \leq \frac{2^{2\lambda}}{p} + \frac{q^2}{2 \cdot p}
\]

where \( p \) is the size of the group and \( q \) is the number of oracle queries.

**Proof.** The proof is given in the GGM (described in Sect. 2.4.2) and shows that the use of leakage does not allow the adversary to learn additional group elements over the game without leakage. Since the adversary can only learn partial information about unknown group elements, it follows that the adversary’s advantage can, at most, be increased by the number of bits that can be learnt about a single element. By showing that each element is only leaked at most two times, the adversary’s advantage can, at most, be (multiplicatively) increased by \( 2^{2 \cdot \lambda} \) over the advantage in the game where no leakage is involved. The advantage of an adversary being able to win the game without leakage is at most \( \frac{1}{p} \), the probability that the adversary can guess the key, since they receive no information about the secret key without leakage.

Group elements will be represented by polynomials, which will be instantiated at the end of the computation. The polynomials allow the game to keep track of which elements the adversary has asked for in a straightforward manner. Since the polynomials are instantiated at the end of the computation, the adversary’s decisions cannot depend on the actual values of the elements.
Instantiation of the polynomials at the end is a common trick used within the literature. However, if two (non equal) polynomials collide, when instantiated, the simulation fails as a single group element now has multiple representations. Thus, it must be shown that the chance of an adversary forcing this collision is small ($\frac{q^2}{2p}$ for $q$ oracle queries).

After showing the bounds for learning the key with leakage, the increased advantage of utilising leakage and the advantage of causing the polynomials to collide, they result in the bound given in the theorem statement. The bound for guessing the key without leakage is $\frac{1}{p}$. The leakage increases this advantage by at most $2^{2\lambda}$, to give an advantage of $\frac{2^{2\lambda}}{p}$. The other way for an adversary to win the game is to cause a collision in the polynomials (which happens with probability $\frac{q^2}{2p}$). Combining this altogether gives the required bound of $\frac{2^{2\lambda}}{p} + \frac{q^2}{2p}$.

The oracles used by the GGM are given in Fig. B.1 in App. B. Since this appendix contains the generic oracles which are used for schemes which also involve a pairing, the adversary here is only allowed access to the oracles for group 1 and cannot use the pairing oracle. Since this scheme does not define a second group, a pairing is also undefined and therefore the adversary can not utilise these oracles. These oracles will be used later in the chapter when a pairing is required.

Let $K$, $\{R_j\}_{j=0}^{q_U}$, $\{U_j\}_{j=1}^{2q_O}$ be indeterminants where $q_U$ is the number of Update calls and $q_O$ is the number of group oracle calls. Let $q = 2 \cdot q_U + 3 \cdot q_O$, the factor 3 arises from the fact the adversary can guess the representation of two elements input to a binary operation and learns another one from the output, thus adding, at most, 3 elements to the list per oracle call. Each Update call updates two shares, adding two new elements to the list. The indeterminants represent the following: $K$ is the secret key. $\{R_j\}_{j=0}^{q_U}$ are the randomness used to update the key ($R_j$ in the scheme description). There are $q_U + 1$ terms, despite the update function being called $q_U$ times because initially $S_0 = R_0$. Finally $\{U_j\}_{j=1}^{2q_O}$ represent any elements that are guessed in $G_1$. The list $L$ is used to keep track of polynomials and their representation. It initially contains the following:

$$L = \{(1, \xi^1_1)\} \cup \{(R_j, \xi^1_{j+2})\}_{j=0}^{q_U}$$
where the $\xi_j^1$ are chosen uniformly at random from $\Xi^1$, such that no two polynomials have the same representation. The list contains the generator of the group and the randomness used to update the key. Since the adversary does not have access to this list, precomputing these elements does not change the adversary’s advantage when playing this game.

At the end of the experiment, the Adversary $A$ outputs a group element $F$ and is said to have won if:

1. $F_i = F_j$ for $i \neq j$

2. $K - F = 0$

The first case corresponds to the adversary being able to create two distinct polynomials which evaluate to the same value, at the end of the computation. This means that a group element has two different representations, breaking the simulation. The second case corresponds to the adversary outputting the correct key (the adversary outputs a polynomial $F$ which equals the key). This corresponds to the adversary winning the CKR-UL game. We will first bound the first case, before going on to bound the second case.

All polynomials are of degree one initially (by construction), and there is no operation which increases the degree. Therefore, the probability of two polynomials evaluating to the same value is $\frac{1}{p}$ by the Schwartz-Zippel lemma. Since there are at most $q$ polynomials (update creates $2 \cdot qU$ polynomials and a call to a GGM oracle can create, at most, $3 \cdot qO$ polynomials), there are \( \binom{q}{2} \leq \frac{q^2}{2} \) pairs of polynomials that could collide and thus the probability of any two polynomials colliding is $\frac{q^2}{2p}$.

Due to the OCLI assumption, both $U^\diamondsuit$ and $U^\heartsuit$ have access to different secret elements which can be leaked upon and because of the randomness used to update the key, each iteration will have access to different secret information. The lists $L_i^\diamondsuit$ and $L_i^\heartsuit$ will represent the set of all elements which can be calculated by $l_i^\diamondsuit$ and $l_i^\heartsuit$ respectively.

The adversary will win if they can repeatedly get the key into the leakage function to reveal it. If the adversary can only get the key into a single leakage function they will not be able to learn the entire secret key, since the leakage per call must be less than the size a group element. Thus the adversary must be able to access it in multiple leakage functions.
Let $\mathcal{L}_i^\oplus$ be the set of elements that could be computed by the leakage function $l_i^\oplus$. Based on the leakage set in the theorem we have that:

$$\mathcal{L}_i^\oplus = \{a \cdot S_i^\oplus + b \cdot R_{i+1} + C\}$$

Where $a, b \in \mathbb{F}_p$ and $C \in \mathbb{F}_p[\{U_j\}_{j=1}^{2^q}]$ and $S_i^\oplus = \sum_{j=0}^{i} R_j$, to correspond to the definition of $S_i^\oplus$ used within the key update mechanism.

Let $\mathcal{L}_i^\oplus$ be the set of elements that could be computed by the leakage function $l_i^\oplus$. Therefore:

$$\mathcal{L}_i^\oplus = \{a \cdot S_i^\oplus + b \cdot R_{i+1} + C\}$$

Where $S_i^\oplus = \mathcal{K} - \sum_{j=0}^{i} R_j$.

From these sets, it can be seen that there are no linear combinations of elements in the leakage which allow the adversary to leak directly upon the secret key $\mathcal{K}$: it always appears with at least a second unknown. Below we show that each unknown cannot be leaked upon enough times for the adversary to learn them.

To bound the amount of leakage an adversary can learn, only leakage functions that contain an unknown group element will be considered. Since, if all components, involved in a leakage function, are completely known by the adversary, the adversary will gain no information from leaking on them. All that is left to show is that no unknown element can be leaked upon more than a bounded number of times (this can be shown to be 2 times):

- $S_i^\oplus$ can be leaked on twice; once in $\mathcal{L}_{i-1}^\oplus$ since $S_{i-1}^\oplus$ is input and $R_i^\oplus$ is generated internally (represented by the polynomials $S_{i-1}^\oplus$ and $R_i^\oplus$), and once in $\mathcal{L}_i^\oplus$ since it is input.
- $S_i^\oplus$ can be leaked on twice; once each in $\mathcal{L}_{i-1}^\oplus$ and $\mathcal{L}_i^\oplus$ due to a similar argument as above.
- $R_{i+1}$ can be leaked on twice; once each in $\mathcal{L}_{i-1}^\oplus$ and $\mathcal{L}_i^\oplus$ since it is generated in $U^\oplus$ and then passed into $U^\oplus$ on the $i^{th}$ iteration.

Since each element can only be leaked on at most twice, the adversary can only learn up to $2 \cdot \lambda$ bits of information per secret. Therefore, the adversary's advantage can be at most $2^{2^2}$
times the advantage of playing the non-leakage variation. Thus, resulting in the bound as stated in the theorem.

This shows that the KP key update mechanism $KU$ is $L_{update}$-CKR-UL secure. This security requirement is necessary for the security of a scheme when leakage is involved. Therefore, in the remainder of the chapter we will be able construct secure schemes on top of $KU$. We now present an attack on the $L_{update}$-DKR-UL security of $KU$. This shows that it does not meet the stronger notion and therefore it may not be possible for security with leakage to decompose to security without leakage and the leakage security of the key update mechanism.

**Proposition 2.** In the GGM, there exists an adversary $A$ who can win the $L_{update}$-DKR-UL game against $KU$ with probability $1 - 2^{-\lambda}$, where $\lambda$ is the number of bits allowed to leak per leakage function (to $U^{\circ\circ}$ and $U^{\circ\circ}$).

**Proof.** The proof is given for the single query case. Multiple queries can be used to increase the adversary’s advantage. The adversary $A$ is given in Fig. 5.11. $\cdot |\lambda$ denotes taking the first $\lambda$ bits of the input. If the key given to the adversary is $k_0$ then the adversary returns the correct answer with probability 1, since $k_0 \cdot (S_i^{\circ\circ})^{-1} = S_i^{\circ\circ}$ because it was $k_0$ that was used to create the shares such that $k_0 = S_i^{\circ\circ} \cdot S_i^{\circ\circ}$. If the adversary was given $k_1$, $l_i^{\circ\circ}(S_i^{\circ\circ}, R_{i+1}^{\circ\circ}) = k_1 \cdot (S_i^{\circ\circ})^{-1} = k_1 \cdot k_0^{-1} \cdot S_i^{\circ\circ}$ which, due to the GGM, is a random string of length $\lambda$. Hence, the probability that $\Lambda_i^{\circ\circ}$ and $\Lambda_i^{\circ\circ}$ are equal (meaning the adversary returns the wrong answer) is $2^{-\lambda}$. This gives the result required.

Figure 5.11.: An adversary against the $L_{update}$-DKR-UL security of $KU$

Proof. The proof is given for the single query case. Multiple queries can be used to increase the adversary’s advantage. The adversary $A$ is given in Fig. 5.11. $\cdot |\lambda$ denotes taking the first $\lambda$ bits of the input. If the key given to the adversary is $k_0$ then the adversary returns the correct answer with probability 1, since $k_0 \cdot (S_i^{\circ\circ})^{-1} = S_i^{\circ\circ}$ because it was $k_0$ that was used to create the shares such that $k_0 = S_i^{\circ\circ} \cdot S_i^{\circ\circ}$. If the adversary was given $k_1$, $l_i^{\circ\circ}(S_i^{\circ\circ}, R_{i+1}^{\circ\circ}) = k_1 \cdot (S_i^{\circ\circ})^{-1} = k_1 \cdot k_0^{-1} \cdot S_i^{\circ\circ}$ which, due to the GGM, is a random string of length $\lambda$. Hence, the probability that $\Lambda_i^{\circ\circ}$ and $\Lambda_i^{\circ\circ}$ are equal (meaning the adversary returns the wrong answer) is $2^{-\lambda}$. This gives the result required.
It is possible to give a weaker definition of DKR-UL which is not susceptible to this type of attack. In the weaker notion the adversary does not see the challenge key until they have made all of their leakage queries. A discussion of when the adversary is allowed leakage can be found (in the context of encryption) in Sect. 5.5.2. Since this weaker notion does not provide the desirable reduction properties of DKR-UL, we do not discuss it any further here.

We leave it as an open problem to construct a key update mechanism which is \( L_{update}^{DKR-UL} \) secure.

5.4. A Leakage Resilient Message Authentication Code

This work was jointly carried out with Professor Elisabeth Oswald, Dr Martijn Stam and Dr Marcin Wójcik [115]. I worked on the model, the scheme, proving of security of the MAC following discussions with both Martijn and Elisabeth, and provided a baseline for the code which was implemented and analysed by Marcin.

Using the techniques discussed above we create a MAC which is leakage resilient in the GGM. We first define our MAC and show that it meets the standard sEUF-CMA notions of security. Next we discuss the modelling of leakage for a MAC, which is more interesting for symmetric primitives than it is for public-key primitives, and show that our scheme is secure within this model. We conclude this section with an alternate construction for a MAC which only meets the weaker notion of EUF-CMA security but is more efficient and can support more leakage.

5.4.1. A MAC Without Leakage

In this section we will introduce our MAC scheme and show that it is sEUF-CMA secure. Before the MAC is introduced, a new Bilinear Diffie-Hellman problem is required, which will be used in the reduction for the MAC security. The new Diffie-Hellman problem will be shown to have its security ‘sandwiched’ between two well known, and well studied, Diffie-Hellman problems. A variation of the problem is then introduced which allows the proof reduction to be given in a cleaner manner without changing the resulting security of the scheme.

**Definition 34** (Target Bilinear Diffie-Hellman Problem (TBDH)). Let \( \mathbb{B}G = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, e, p) \)
be a set of groups with a pairing between them. The target bilinear Diffie-Hellman problem is then defined as; given \( g_1, g_2, g_2^x, g_3^y \) find \( g_3^{x \cdot y} \), where \( x, y \) are chosen uniformly at random from \( \mathbb{Z}_p \). Given an adversary \( A \), their advantage is defined as:

\[
\text{Adv}_{\text{TBDH}}(A) = \Pr[A(g_1, g_2, g_2^x, g_3^y) = g_3^{x \cdot y}].
\]

If the discrete logarithm problem (Def. 6) is easy in either \( G_2 \) or \( G_3 \) then the TBDH problem is also easy. An adversary for the TBDH problem can be used to solve the CBDH problem (Def. 9) and since the CBDH problem is believed to be hard this gives credence to the TBDH assumption.

**Theorem 5.** Given a bilinear group \( \mathbb{B}G = (G_1, G_2, G_3, e, p) \), let \( A \) be an adversary against the TBDH problem for \( \mathbb{B}G \), then there exists an adversary \( B \) (of similar complexity to \( A \)) against the CBDH problem for \( \mathbb{B}G \) such that:

\[
\text{Adv}_{\text{TBDH}}(A) \leq \text{Adv}_{\text{CBDH}}(B).
\]

**Proof.** Let \( A \) be an adversary against the TBDH problem for \( \mathbb{B}G \) then Fig. 5.12 shows how to construct an adversary \( B \) against the CBDH problem. From this reduction it can be seen that \( B \) will win whenever \( A \) does, giving \( \text{Adv}_{\text{TBDH}}(A) \leq \text{Adv}_{\text{CBDH}}(B) \) as required. \( \square \)

\[
\text{adversary} \ B(g_1, g_2, g_2^x, g_3^y):
\]

\[
g_3^y \leftarrow e(g_1, g_2^y)
\]

\[
f \leftarrow A(g_1, g_2, g_2^x, g_3^y)
\]

Return \( f \)

Figure 5.12.: Constructing a CBDH Adversary from a TBDH Adversary

We now introduce the Target Bilinear Diffie-Hellman with Oracle Problem (TBDHwO), which is the same as the TBDH problem with an additional oracle \( test(\cdot) \) which takes a group element and checks if it is equal to \( g_3^{x \cdot y} \). The TBDHwO problem will be used in the proof for the MAC, since it makes the reduction cleaner without weakening the security statement.
**Definition 35** (Target Bilinear Diffie-Hellman with Oracle Problem (TBDHwO)). Let $\mathbb{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, e, p)$ be a set of groups with a pairing between them. The target bilinear Diffie-Hellman problem is then defined as; given $g_1, g_2, g_3^x, g_3^y$, and access to an oracle $test(\cdot)$ which determines if a given element equals $g_3^{x\cdot y}$, find $g_3^{x\cdot y}$, where $x, y$ are chosen uniformly at random from $\mathbb{Z}_p$. Given an adversary $A$, their advantage is defined as:

$$Adv_{\mathbb{G}}^{\text{tbdh}}(A) = Pr[f = g_3^{x\cdot y} : A^{\text{test}(\cdot)}(g_1, g_2, g_3^x, g_3^y) = f].$$

**Lemma 2.** Given a bilinear group $\mathbb{G} = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, e, p)$, let $A$ be an adversary against the TBDHwO problem for $\mathbb{G}$, then there exists an adversary $B$ (of similar complexity to $A$) against the TBDH problem for $\mathbb{G}$ such that:

$$Adv_{\mathbb{G}}^{\text{tbdhwo}}(A) \leq (q_t + 1) \cdot Adv_{\mathbb{G}}^{\text{tbdh}}(B)$$

where $q_t$ is the number of times that the test oracle is called.

**Proof.** Let $A$ be an adversary against the TBDHwO problem, then it is possible to construct an adversary $B$ that breaks the TBDH problem. Without loss of generality, we assume that $A$ never makes the same call to the test oracle twice and the value output by $A$ is passed to the test oracle before being output (hence the plus one in the security bound, to add this query if it was not previously made). It is clear that if $B$ outputs the same answer as $A$, then $B$ will win whenever $A$ wins. However, $B$ cannot simulate the test oracle. If $A$ makes $q_t$ test queries, $B$ can select one of these uniformly at random and return this value as their answer, while returning false for the remaining test queries. If the adversary $B$ chooses the correct test query, with probability $\frac{1}{q_t + 1}$ they will win the game whenever $A$ does, giving the desired result. \hfill $\Box$

Now that all the preliminary assumptions are in place, the MAC scheme can be presented and shown to provide sEUF-CMA security. Let $H : \{0, 1\}^* \rightarrow \mathbb{G}_2$ be a hash function, then the MAC $M = (KG, Tag, VRFY)$ is given in Fig. 5.13.

The scheme can be understood as follows; key generation selects a random element in $\mathbb{G}_1$ to be the secret key. Tag generation first hashes the message, then pairs the secret key and the hash
\textbf{proc KG:}\n\[ k \leftarrow G_1 \]
\textbf{Return} $k$

\textbf{proc Tag}(k, M):
\[ W \leftarrow H(M) \]
\[ \sigma \leftarrow e(k, W) \]
\textbf{Return} $\sigma$

\textbf{proc VRFY}(k, M, $\sigma$):
\[ W \leftarrow H(M) \]
\[ \sigma' \leftarrow e(k, W) \]
\textbf{Return} $\sigma' = \sigma$

**Figure 5.13.** Our bilinear MAC scheme

\textbf{adversary} $B(\mathcal{H} = g_2^x, \mathcal{F} = g_3^y)$:
\[ i \leftarrow 0 \]
\[ j \leftarrow [q_h] \]
\[ (M, \sigma) \leftarrow A^{H(\cdot), \text{Tag}(\cdot), \text{Verify}(\cdot, \cdot)}() \]
\textbf{Return} $M$

\textbf{simulator} $H(m)$:
\[ i \leftarrow i + 1 \]
\textbf{if} $W[m] = \perp$ \textbf{then}
\[ j \leftarrow i \]
\textbf{Return} $\mathcal{H}$
\textbf{else}
\[ W[m] \leftarrow \mathbb{Z}_p \]
\textbf{end if}
\textbf{end if}
\textbf{Return} $g_2^m$

\textbf{simulator} Tag$(M)$:
\textbf{if} $W[M] = \perp$ \textbf{then}
\[ W[M] \leftarrow \mathbb{Z}_p \]
\textbf{else if} $W[M] = \times$ \textbf{then}
\[ \text{ABORT} \]
\textbf{end if}
\[ \sigma \leftarrow \mathcal{F}^{W[M]} \]
\textbf{Return} $\sigma$

\textbf{simulator} Verify$(M, \sigma)$:
\textbf{if} $W[M] = \perp$ \textbf{then}
\[ W[M] \leftarrow \mathbb{Z}_p \]
\textbf{else if} $W[M] = \times$ \textbf{then}
\[ \text{Return test}(\sigma) \]
\textbf{end if}
\[ \text{Return} (\sigma = \mathcal{F}^{W[M]}) \]

**Figure 5.14.** Constructing a TBDHwO Adversary from a sEUF-CMA Adversary

of the message, using a bilinear map, to create the tag. Verification takes the message and a tag and checks the tag’s validity by using the message to construct the correct tag and comparing it to the one provided (true corresponds to 1 and false to 0). Theorem 6 gives the sEUF-CMA (Def. 27) security for the MAC.

**Theorem 6.** Let $H : \{0, 1\}^* \rightarrow G_2$ be modelled as a random oracle and $A$ be a sEUF-CMA adversary against $M$ who makes $q_h$ queries to the hash function and $q_v$ verification queries, then there exists an adversary $B$ (of similar computational complexity) against the TBDH problem such that:

$$\text{Adv}^{\text{sEUF-CMA}}_M(A) \leq (q_h + 1) \cdot (q_v + 1) \cdot \text{Adv}^{\text{BDH}}(B).$$
Proof. Let $A$ be an adversary against the sEUF-CMA security of $M$, then Fig. 5.14 shows how to construct an adversary $B$, who uses $A$ to solve the TBDHwO problem.

Without loss of generality, we can assume that, if $A$ creates a forgery $\sigma$, on $M$, they queried $H(M)$. If this is not the case, it is possible to construct an equivalent adversary who performs the same actions but hashes $M$ before returning their forgery, at the cost of an extra hash query. The reduction works by simulating the key $gy_1$ by using the knowledge of $gy_3$. If the adversary can create a valid forgery on a message $M$ which hashes to $gy_2$, the tag will be $g^x_3y$ leading to $B$ being able to win the TBDHwO game. Thus, whenever $A$ wins the sEUF-CMA security game with a message that hashes to $gy_2$, $B$ solves the TBDHwO problem.

The reduction works by embedding $gy_2$ into the hash function randomly (for some position $j$) for one of the $qh + 1$ random oracle calls made. The probability that the message forged on is at position $j$ is $\frac{1}{qh+1}$. The ABORT does not affect this probability since this is only triggered when the adversary requests the tag of the message with hash value $gy_2$, at which point the adversary can no longer create a forgery on this message. Thus, $Adv_{seufcma}^M(A) \leq (qh + 1) \cdot Adv_{BG}^{TBDHwO}(B)$ and combining with Lemma 2 gives the desired result. 

5.4.2. Model

In this section we discuss how to take the sEUF-CMA security game (Def. 27) and add leakage to it. The key generation algorithm will not be able to leak, as this would allow leakage on the underlying secret key, instead of just the shares. This matches the decisions made within previous work [75, 101] and aligns with practice. In reality, a device would have the key generated and embedded into it during production and thus the adversary would not be able to receive leakage during this phase.

This leaves us to consider what should be allowed to leak; Tag, Verify or both. This is the first time that such a question has arisen, since all previous work has been within the public key setting and thus, ignoring key generation for the reasons explained above, there is only a single algorithm which processes any secret information, making the decision trivial. The ideal would be to allow both Tag and Verify to leak since it allows as much leakage as possible. However, for reasons explained below, we consider the situation where only Tag is allowed to
Below we give a security notion, involving sets of leakage functions, which captures all three of these notions. A set of leakage functions for Tag or Verify containing, a single, constant function (without loss of generality we will use the zero function denoted \{0\}) is equivalent to not allowing the oracle to leak (since it provides no information). Having both sets of leakage functions be the constant function, results in the original sEUF-CMA definition.

Making Verify leak is problematic, in both our construction and any constructions that follow the (fairly standard) design paradigm of having Verify reconstruct the tag and compare it to the provided tag. Since the leakage is adaptive, the following attack arises (for certain classes of leakage); the adversary can take a message and choose a random value to be the “tag” before submitting both of these to Verify. Since Verify has to compute the real tag, the adversary can then leak on this. By repeating this process multiple times (depending on the amount of leakage allowed per call) the adversary can completely learn the real tag. The tag and message pair can then be submitted as a forgery as this message was never queried to the Tag oracle. We leave it as an open question as to how to construct a scheme with a verify function which is allowed to leak.

**Definition 36** (Strong Existential Unforgeability Under Chosen Message with Leakage Attack (sEUF-CMLA)). Let \(M^* = (KG^*, \text{Tag}^*, \text{VRFY}^*)\) be a MAC. Then Fig. 5.15 defines the \((\mathcal{L}_{\text{tag}}, \mathcal{L}_{\text{vrfy}})\)-sEUF-CMLA security game. The advantage of an adversary \(A\) winning the game is defined as \(\text{Adv}_{M^*}^{(\mathcal{L}_{\text{tag}}, \mathcal{L}_{\text{vrfy}})\text{-seufcmla}}(A) = \text{Pr}[\text{Exp}_{M^*}^{(\mathcal{L}_{\text{tag}}, \mathcal{L}_{\text{vrfy}})\text{-seufcmla}}(A) = 1]\).

It is worth noting that the \((\mathcal{L}_{\text{tag}}, \mathcal{L}_{\text{vrfy}})\)-sEUF-CMLA leakage game above is defined for arbitrary sets of leakage functions. Our scheme will use the OCLI assumption to prove security...
using the set of leakage functions $L_{\text{tag}}$ defined in Sect. 5.1. The key generation function $KG^*$ will then output $(S_{G#0}, S_{G#0})$ and the leakage function is a tuple containing the two leakage functions $l_{G#}, l_{G#}$, such that they obey the OCLI assumption. Finally, $Tag^*$ on input $(S_{G#i}, S_{G#i})$ will run $Tag_{G#}$ followed by $Tag_{G#}$ (as defined by the scheme) passing the information between them as detailed previously.

EUF-CMLA can be defined in a similar manner to sEUF-CMLA above, except the forgery must be on a new message not just a new message/tag pair (as in the non-leakage scenario).

### 5.4.3. A Leakage Resilient MAC

Now that the leakage model is in place, the MAC $M$ can be adjusted such that it is secure in the face of leakage on any tag queries (the set $L_{\text{vrfy}}$ contains the zero function). The leakage resilient MAC $M^*$ is then given in Fig. 5.16.

The MAC $M^*$ was constructed using the definition of key update splittable for the KP key update mechanism discussed previously.

**Claim 2.** The $Tag$ function for MAC $M$ is key update splittable conforming to the KP key update mechanism $KU$.

**Proof.** The MAC $M$ can be converted into the leakage resilient MAC $M^*$ (Fig. 5.16). Below we
show that the tag output from the shared function $\text{Tag}^*$ is identical to that output by the original $\text{Tag}$:

$$\text{Tag}^*((S_1^\cdot, S_1^\cdot), M) = \sigma_i$$

$$= \sigma_i^\cdot \cdot \sigma_i^\cdot$$

$$= e(S_1^\cdot, H(M)) \cdot e(S_1^\cdot, H(M))$$

$$= e(S_1^\cdot \cdot S_1^\cdot, H(M))$$

$$= e(k, H(M))$$

$$= \text{Tag}(k, M).$$

Due to the above, and the fact $\text{Tag}^\cdot$, $\text{Tag}^\cdot$ implicitly calls $U^\cdot$, $U^\cdot$ respectively (performs same operations) to update the key (meaning the shares will have the same distribution), $M$ is key update splittable.

Since $M^*$ is functionally equivalent to the MAC $M$ without leakage, its sEUF-CMA security follows. Now all that is left to show is that it is secure when leakage is allowed on the tagging oracle.

**Theorem 7.** Let $A$ be an adversary against the $(\mathcal{L}_{\text{tag}}, \{0\})$-sEUF-CMLA security of $M^*$ in the GGM, then there exists an adversary $B$ (of similar complexity to $A$) against the sEUF-CMA security of $M$ such that:

$$\text{Adv}_{M^*}^{(\mathcal{L}_{\text{tag}}, \{0\})-\text{seufcmla}}(A) \leq 2^{4\lambda} \cdot \text{Adv}_M^{\text{seufcma}}(B) + \frac{q^2}{p}$$

where $q$ relates to the number of queries made to the oracles, $p$ is the size of the group and $\lambda$ is the amount of leakage allowed (for each leakage function to $\text{Tag}^\cdot$ and $\text{Tag}^\cdot$).

**Proof.** The proof is given in the GGM and will follow a similar structure to that of Theorem 4. By showing that the use of leakage does not allow the adversary to learn any elements that they would be unable to learn if no leakage had been involved, it follows that the adversary’s advan-
tage can, at most, be increased by the number of bits learnt about a single element. This follows from how the GGM behaves; since each element is represented by a random (and independent) bitstring, information about one group element gives no information about any others. By showing that each element is only leaked at most four times, the adversary’s advantage is, at most, increased, multiplicatively, by $2^{4\cdot\lambda}$ over the advantage in the game where no leakage is involved. In Theorem 4 it was shown that each element can only be leaked on twice. This is still true for the key shares. However, it is possible to leak on other elements computed by the MAC, up to, four times.

Group elements will be represented by polynomials, which will be instantiated at the end of the computation. The polynomials allow the game to keep track of which elements the adversary has asked for in a straightforward manner and because they are instantiated at the end of the computation, the adversary’s decisions clearly cannot depend on the actual values of the elements. As before, it must be shown that the chance of these polynomials colliding when evaluated is small (it will be shown to be $\frac{q^2}{p}$). Since the MAC involves three groups, polynomials will be tracked per group.

The oracles used by the GGM are given in Fig. B.1 in App. B.

Let $\mathcal{K}, \{R_j\}_{j=0}^{q_T}, \{H_j\}_{j=1}^{q_H}, \{U_j\}_{j=1}^{2q_O}, \{V_j\}_{j=1}^{2q_O}, \{W_j\}_{j=1}^{2q_O}$ be indeterminants where $q_H$ is the number of hash queries, $q_T$ is the number of Tag calls and $q_O$ is the number of group oracle calls. Let $q = q_H + 5 \cdot q_T + 3 \cdot q_O$, the factor 3 on the oracle arises from the fact the adversary can guess the representation of two elements going into a binary operation and learns another one from the output, thus adding at most 3 elements to the list per oracle call. Five new elements are added due to a tagging call; the two updated shares, the tag and the partial tags $\sigma^G, \sigma^G$. The indeterminants represent the following: $\mathcal{K}$ is the secret key, $\{R_j\}_{j=0}^{q_T}$ are the randomness (the $R_i$’s) used to update the key, $\{H_j\}_{j=1}^{q_H}$ represent any hash function queries and $\{U_j\}_{j=1}^{2q_O}, \{V_j\}_{j=1}^{2q_O}, \{W_j\}_{j=1}^{2q_O}$ represent any elements that are guessed in $G_1, G_2, G_3$ respectively. The lists $L_1, L_2, L_3$ are used to keep track of polynomials and their representations in $G_1, G_2, G_3$ respectively. In Theorem 4 only a single list was required, as only a single group
was utilised. Here we have a list per group. They are initialised as follows:

\[
\mathcal{L}_1 = \{(1, \xi^1_1)\} \cup \{(\mathcal{R}_j, \xi^1_{j+2})\}_{j=0}^{q_T} \\
\mathcal{L}_2 = \{(1, \xi^2_1)\} \\
\mathcal{L}_3 = \{(1, \xi^3_1)\}
\]

where the \(\xi^i_j\) are chosen uniformly at random from \(\Xi^i\), such that no two polynomials have the same representation. The sets the representations are drawn from \(\Xi^1, \Xi^2, \Xi^3\) (for \(G_1, G_2, G_3\) respectively) are all disjoint. All three lists are initialised with the generator of the respective group. Note that it is not strictly necessary to instantiate the generator in \(G_3\) since it can be calculated using the other information provided. We precompute the representations of the randomness used for the key update, the \(R_i\)'s, but since the adversary does not have access to this list of elements, this does not affect the game.

The adversary \(A\) outputs the pair \((M, \sigma)\) and is said to have won if:

1. \(f^l_i = f^l_j\) for \(l \in \{1, 2, 3\}\) and \(i \neq j\)
2. \(\mathcal{K} \cdot H^* - T = 0\) where \(H^*\) is the indeterminant corresponding to the hash of \(M\), \(T\) is the corresponding polynomial for \(\sigma\) and \(\sigma\) was not output from the tag oracle.

The first case corresponds to the adversary being able to create two polynomials which evaluate to the same value. Two distinct polynomials evaluating to the same value, means that a single group element has two distinct representations. This breaks the simulation and therefore the adversary is said to have won the game. The second case corresponds to the adversary being able to create a forgery on the MAC. The polynomial given in the second case evaluating to zero implies that the adversary output a valid forgery for the MAC. Therefore, they have won the sEUF-CMLA game. We will first bound the situation corresponding to a collision, before bounding the second condition which corresponds to winning the game.

All polynomials originally in \(\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3\) are of degree one and the only operation that increases the degree is the pairing operation which can only be called on elements in \(G_1, G_2, G_3\).
Hence $G_3$ can contain degree two polynomials. There are no operations to take an element from $G_3$ to $G_1$ or $G_2$ and thus only elements in $G_3$ can be of degree 2. Hence, by the Schwartz-Zippel lemma, the probability of two (non-zero) polynomials evaluating to the same value is $\frac{2^q}{p}$. Since there are at most $q$ polynomials, there are $\binom{q}{2} \leq \frac{q^2}{2}$ pairs of polynomials that could collide and thus the probability of any two polynomials colliding is $\frac{q^2}{p}$.

We will now bound the probability of the adversary using the leakage to create a forgery. Without loss of generality, we will now only look at leakage in the target group $G_3$. Any element from $G_1$, $G_2$ calculated by the leakage can be transferred over to $G_3$ using a pairing, with the corresponding generator. Any elements known to the adversary can easily be embedded into the leakage by the adversary as required. Therefore, any elements which can be calculated (and leaked upon) in $L_1$ or $L_2$, can be calculated in $L_3$. Hence, this does not weaken the adversary.

Therefore only the leakage from $L_3$ needs to be considered. However, due to the OCLI assumption both $Tag^G$ and $Tag^G$ will have access to different secret information and due to the randomness used to update the key, each iteration will have access to different secret information. The lists $L_i^G$ and $L_i^G$ will represent all elements which can be calculated by $l_i^G$ and $l_i^G$ respectively.

The adversary will win if they can repeatedly get a (new) tag into the leakage function to reveal it. If they can only leak on a forgery once, they will not be able to leak the complete tag and thus they are required to get it into multiple leakage sets, to completely reveal the forgery.

Let $L_i^G$ be the set of elements that could be computed by the leakage function $l_i^G$. Based on the leakage set in the theorem we have that:

$$L_i^G = \{ A \cdot S_i^G + B \cdot R_{i+1} + C \}$$

Where $A, B \in \mathbb{F}_p \{ \{ g_j \}_{j=1}^{q+1}, \{ m_j \}_{j=1}^{k+1} \}$ and $C \in \mathbb{F}_p \{ \{ u_j \}_{j=1}^{q+1}, \{ m_j \}_{j=1}^{k+1} \}$ and $S_i^G$ denotes $\sum_{j=0}^i R_j$.

Let $\hat{L}_i^G$ be the set of elements that could be computed by the leakage function $\hat{l}_i^G$. Therefore:

$$\hat{L}_i^G = \{ A \cdot \hat{S}_i^G + B \cdot \hat{R}_{i+1} + C + d \cdot \hat{S}_i^G \cdot \hat{g}_i \}$$

141
Where \( d \in \mathbb{F}_p \) and \( S_i \) denotes \( \mathcal{K} - \sum_{j=0}^{i} R_j \). Without loss of generality, we will assume that \( i \)th tag call maps to \( \mathcal{H}_i \).

The only tags that can be contained in \( \mathcal{L}_i \) are tags of the form \( \mathcal{H}_j \cdot \mathcal{K} \) for \( j < i \) in which the adversary has included in the leakage and thus there is no advantage in leaking upon \( \mathcal{H}_j \cdot \mathcal{K} \).

Ignoring this trivial case, there are no linear combinations possible that will reveal an unknown tag or the key itself. Similarly, for \( \mathcal{L}_i \), the only tags that can be included are of the form \( \mathcal{H}_j \cdot \mathcal{K} \) for \( j \leq i \). This is because the adversary can again ask to leak on tags they have already seen by embedding them in the leakage function but this time can also leak on \( \mathcal{H}_i \cdot \mathcal{K} \). This is also not of any use because the leakage on this tag will be received at the same time the adversary is given the tag and thus no extra information is gained.

To bound the leakage an adversary receives, we will only consider leakage on elements that operate on an unknown group element. If all elements are completely known by the adversary, the adversary will gain no information from leaking on them. All that is left to show is that no element can be leaked upon more than a bounded number of times (it can be shown to be at most 4 times):

- \( S_i \) can be leaked on twice; once in \( \mathcal{L}_{i-1} \) since \( S_{i-1} \) is passed in and \( R_i \) is generated internally (represented by the polynomials \( S_{i-1} \) and \( R_i \)), and once in \( \mathcal{L}_i \) since it is passed in.
- \( S_i \) can be leaked on twice; once each in \( \mathcal{L}_{i-1} \) and \( \mathcal{L}_i \) due to a similar argument as above.
- \( R_{i+1} \) can be leaked on twice; once each in \( \mathcal{L}_{i-1} \) and \( \mathcal{L}_i \) since it is generated in \( \mathcal{U} \) and then passed into \( \mathcal{U} \) on the \( i \)th iteration.
- \( S_i \cdot \mathcal{H}_i \), the intermediated state, can be leaked on 4 times; once in each of \( \mathcal{L}_{i-1}, \mathcal{L}_i, \mathcal{L}_{i-1}, \mathcal{L}_i \) using the argument above for calculating the next share and the fact for message \( M_i \), on the \( i \)th tag call the intermediate state is generated in \( \text{Tag} \) and passed into \( \text{Tag} \). To leak on this four times will require tagging the same message twice.

Since each element can only be leaked on, at most, four times, the adversary can only learn up to \( 4 \cdot \lambda \) bits of information per secret. Therefore, the adversary’s advantage can be at most
$2^{4\cdot \lambda}$ times the advantage of playing the standard non-leakage game. Putting this together with the probability of the polynomials colliding, gives the bound in the theorem.

From this, it can be seen that the security is related to the number of queries allowed in a quadratic manner and the amount of leakage allowed in an exponential manner. For a high level intuition, if we ignore the generic group model queries, if a security level of $2^{-n}$ is required and the underlying MAC has security $2^{-m}$ then $\lambda = O\left(\frac{m-n}{4}\right)$.

While it would be desirable to be able to have some leakage on Verify, there are certain situations where allowing leakage on only the tagging algorithm should be enough. For example, the Intel Software Guard Extension (SGX), an extension designed to provide integrity and privacy of computation [46], has a MAC inbuilt which, as soon as a verify is failed, the process aborts and the device stops working until the keys are changed [86]. The proof above would be able to be adapted to cases where the adversary could get leakage on successful verifications and a single failed verification query. The proof for the successful verification queries’ leakage would be similar to a tag leakage query except that the adversary would already know the tag at the start of the query, instead of learning it at the end. The failed verification query would give the adversary some extra information but not enough to construct a forgery. Hence the MAC can still be proven secure as used by Intel SGX.

### 5.4.4. An Alternative MAC

In this section we provide an alternative MAC construction. The advantage of this MAC is twofold. Firstly, the tagging is more efficient; only a single pairing operation is used within the tagging operation. When the scheme is made leakage resilient the number of pairings does not increase, at the expense of adding a second pairing to the verify operation. Secondly, this scheme can support more leakage per tagging operation than the previous scheme. These advantages come at the expense of only providing EUF-CMA security instead of the more desirable sEUF-CMA security. It is only guaranteed to be hard to create a tag forgery on a message not seen before, while sEUF-CMA states it is hard to provide a tag that has not been seen before even if it is on the same message. In fact, for this particular scheme, it is trivial to use a tag on a message to create a second tag on the same message (verify checks the relation $e(\sigma_1, H(M)) = \ldots$
Figure 5.17.: Our alternate bilinear MAC scheme $M_1$

$s_2 \cdot e(k, H(M))$ and thus $(s_1, s_2)$ can be adjusted so this relationship still holds. Making schemes more efficient in the face of leakage has been considered before; by considering leakage resilient PRFs where the inputs must be chosen non-adaptively, Faust et al. [67] are able to create a scheme which is more efficient than other (adaptive) PRFs in the literature [56]. Unfortunately, it is still an open question as to how to allow leakage on verify. An attack involving leaking on the (deterministic) unknowns still arises (similar to the one on the previous MAC); using repeated leakage queries the adversary can learn $e(k, H(M))$ and using this can generate a valid forgery.

Figure 5.17 gives the new MAC $M_1$ which is secure without leakage. Security is shown via a reduction; if an adversary can create a forgery, therefore breaking the EUF-CMA security (Def. 27) of $M_1$, they can be used to construct an adversary who breaks the EUF-CMA security of $M$. This implies that the sEUF-CMA security of $M$ can also be broken which contradicts Theorem 6. The following theorem formalises this intuition.

**Theorem 8.** Let $A$ be a EUF-CMA adversary against $M_1$, then there exists an adversary $B$ (of similar computational complexity) against the EUF-CMA security of $M$ such that:

\[
\text{Adv}_{M_1}^{\text{eufcma}}(A) \leq \text{Adv}_{M}^{\text{eufcma}}(B).
\]

**Proof.** Let $A$ be an adversary against the EUF-CMA security of $M_1$ (with oracle access to Tag’, Verify’) , then it is possible to construct an adversary $B$, who uses $A$ (and calls to their own oracles Tag, Verify) to break the EUF-CMA security of $M$.

Figure 5.18 gives the reduction. The adversary $A$ returns a forgery $M,(\sigma_1, \sigma_2)$ such that $e(\sigma_1, H(M)) = \sigma_2 \cdot e(k, H(M))$. Given the forgery, the adversary $B$ sets $\sigma = e(\sigma_1, H(M))$.

\[
\text{Adv}_{M}^{\text{eufcma}}(B) \leq \text{Adv}_{M_1}^{\text{eufcma}}(A).
\]

144
adversary \( B() \):

\[
(M, \sigma') \leftarrow A^{\text{Tag}()}(\cdot), \text{Verify}'(\cdot, \cdot)()
\]

\[
(\sigma_1, \sigma_2) \leftarrow \sigma'
\]

\[
\sigma \leftarrow e(\sigma_1, H(M)) \cdot \sigma_2^{-1}
\]

Return \((M, \sigma)\)

simulator \( \text{Tag}'(M) \):

\[
\sigma \leftarrow \text{Tag}(M)
\]

\[
\sigma_1 \overset{\$}{\leftarrow} G_1
\]

\[
\sigma_2 \leftarrow e(\sigma_1, H(M)) \cdot \sigma_1^{-1}
\]

\[
\sigma' \leftarrow (\sigma_1, \sigma_2)
\]

Return \(\sigma'\)

simulator \( \text{Verify}'(M, \sigma) \):

\[
(\sigma_1, \sigma_2) \leftarrow \sigma
\]

\[
\sigma' \leftarrow e(\sigma_1, H(M)) \cdot \sigma_2^{-1}
\]

\[
b \leftarrow \text{Verify}(M, \sigma)
\]

Return \(b\)

Figure 5.18.: Constructing a EUF-CMA Adversary Against \( M \) from a EUF-CMA Adversary Against \( M_1 \)

\[
\text{proc KG}^{+}:
\]

\[
k \overset{\$}{\leftarrow} G_1
\]

\[
S_0^i \leftarrow G_1
\]

\[
S_0^i \leftarrow k \cdot (S_0^i)^{-1}
\]

Return \((S_0^i, S_0^i)\)

\[
\text{proc Tag}^+(S_1^i, M):
\]

\[
R \overset{\$}{\leftarrow} G_1
\]

\[
\sigma_1^i \leftarrow S_1^i \cdot R
\]

\[
\sigma_2 \leftarrow e(R, H(M))
\]

\[
R_{i+1}^i \leftarrow G_1
\]

\[
S_{i+1}^i \leftarrow S_1^i \cdot R_{i+1}^i
\]

Return \((S_{i+1}^i, S_1^i, \sigma_1, R_{i+1}^i)\)

\[
\text{proc Tag}^+(S_1^i, \sigma_1, \sigma_2, R_{i+1}^i):
\]

\[
\sigma_1 \leftarrow \sigma_1^i \cdot S_1^i
\]

\[
\sigma_i \leftarrow (\sigma_1, \sigma_2)
\]

\[
S_{i+1}^i \leftarrow S_1^i \cdot (R_{i+1}^i)^{-1}
\]

Return \((S_{i+1}^i, \sigma_i)\)

Figure 5.19.: Alternate Leakage Resilient MAC \( M_1^* \)

\[
\sigma_2^{-1}
\]

and outputs \((M, \sigma)\) as their forgery. Whenever \( M, (\sigma_1, \sigma_2) \) is a valid forgery for \( M_1 \), \( \sigma = e(k, H(M)) \) and therefore \((M, \sigma)\) is a valid forgery against \( M \) since \( B \) never asked for a tag on \( M \), it implies that \( A \) also did not request a tag on \( M \). Thus \( B \) wins the EUF-CMA game against \( M \) whenever \( A \) wins the EUF-CMA game against \( M_1 \), giving the desired result.

Using the KP key update \( \mathcal{KU} \), the MAC \( M_1 \) can be made leakage resilient. Figure 5.19 shows the resulting scheme after it has had the key update mechanism applied to it. The technique used to prove security for \( M_1 \), when leakage is involved, is the same technique used for the original MAC; show that the adversary cannot learn any group elements that cannot be learnt without the use of leakage and therefore the adversary’s advantage is increased at most by the amount of $\Delta 145$.
leakage learnt. For $M^*_1$ the number of times each element can leak is only three times, instead of the previous four and therefore each leakage function is allowed to leak more for the same level of security. This is formalised in the theorem statement below.

**Theorem 9.** Let $A$ be an adversary against the $(\mathcal{L}_{\text{tag}}, \{0\})$-EUF-CMLA security of $M^*_1$ in the GGM, then there exists an adversary $B$ (of similar complexity to $A$) against the EUF-CMA security of $M_1$ such that:

$$\text{Adv}_{M_1}^{(\mathcal{L}_{\text{tag}}, \{0\}) - \text{eufcmla}} (A) \leq 2^{3 \lambda} \cdot \text{Adv}_{M_1}^{\text{eufcma}} (B) + \frac{q^2}{p}$$

where $q$ is the number of queries made to the oracles, $p$ is the size of the group and $\lambda$ is the amount of leakage allowed (to each half of the split of the tagging function per query).

**Proof.** The proof is given in the GGM and is extremely similar to the proof of Theorem 7. The goal is to show that the usage of leakage does not allow the adversary to learn any additional elements. Once this has been proven, the adversary’s advantage can be increased by the number of bits that the adversary can acquire about a single element. This follows from the GGM; information about one element reveals no information about another. By showing that each element can be leaked upon at most three times, the adversary’s advantage is increased by, at most, $2^{3 \lambda}$ over the advantage in the game where no leakage is involved.

As discussed in Sect. 2.4.2 and used in Theorem 7, group elements will be represented by polynomials, which will be instantiated at the end of the game. It will be shown that the chance of an adversary forcing a collision, on the polynomials, when instantiated, is suitably small small \((q^2/p)\).

The oracles used by the GGM are given in Fig. B.1 in App. B.

Let $K_j, \{R_j\}_{j=0}^{qT}, \{X_j\}_{j=1}^{qT}, \{\mathcal{H}_j\}_{j=1}^{qH}, \{\mathcal{U}_j\}_{j=1}^{2qO}, \{\mathcal{V}_j\}_{j=1}^{2qO}, \{\mathcal{W}_j\}_{j=1}^{2qO}$ be indeterminants where $qH$ is the number of hash queries, $qT$ is the number of Tag calls and $qO$ is the number of group oracle calls. The indeterminants represent the following: $K$ is the secret key, \(\{X_j\}_{j=1}^{qT}\) is the randomness used to create the tags, \(\{R_j\}_{j=0}^{qT}\) are the randomness used to update the key, \(\{\mathcal{H}_j\}_{j=1}^{qH}\) represent any hash function queries and \(\{\mathcal{U}_j\}_{j=1}^{2qO}\), \(\{\mathcal{V}_j\}_{j=1}^{2qO}\), \(\{\mathcal{W}_j\}_{j=1}^{2qO}\) represent any elements that are guessed in $G_1, G_2, G_3$ respectively. Let $q = q_H + 5 \cdot q_T + 3 \cdot q_O$, the factor 3 arises from

146
the fact the adversary can guess the representation of two elements being passed into a binary operation and learns another one from the output, thus adding up to 3 elements to the list per oracle call. Each time the tagging oracle is called, five polynomials are added to the lists; the two updated shares, the two elements of the tag and, the polynomial corresponding to, the partial state $\sigma$. The lists $L_1, L_2, L_3$ are used to keep track of polynomials and their representations in $G_1, G_2, G_3$ respectively. They are initialised as follows:

\[
L_1 = \{(1, \xi_1^1)\} \cup \{(\mathcal{R}_j, \xi_{j+2}^1)\}_{j=0}^{q_T} \cup \{(X_j, \xi_{j+q_T+3})\}_{j=0}^{q_T} \\
L_2 = \{(1, \xi_1^2)\} \\
L_3 = \{(1, \xi_1^3)\}
\]

where the $\xi_i^j$ are chosen uniformly at random from $\Xi^i$, such that all polynomials have a unique representation. The sets the representations are drawn from $\Xi^1, \Xi^2, \Xi^3$ (for $G_1, G_2, G_3$ respectively) are all disjoint. All three lists are initially instantiated with the group generator. Note that it is not strictly necessary to instantiate the group generator in $G_3$ since it can be calculated using the other information provided. The representations of the randomness used for the key update, the $R_i$’s, are precomputed, but since the adversary does not have access to this list of elements, this does not effect the game (similarly for the tagging randomness).

The adversary $A$ outputs the pair $M, (\sigma_1, \sigma_2)$ and is said to have won if:

1. $\mathcal{G}_i^l = \mathcal{G}_j^l$ for $l \in \{1, 2, 3\}$ and $i \neq j$

2. $\mathcal{R} \cdot \mathcal{H}^* + \mathcal{T}_2 - \mathcal{T}_1 \cdot \mathcal{H}^* = 0$ where $\mathcal{H}^*$ is the indeterminant corresponding to the hash of $M, (\mathcal{T}_1, \mathcal{T}_2)$ is the corresponding pair of polynomials for $(\sigma_1, \sigma_2)$ and $M$ has previously not been sent to the Tag oracle

The first case corresponds to the adversary being able to create two polynomials which evaluate to the same value. If two polynomials evaluate to the same value, this corresponds to a single group element being given two distinct representations. The adversary is said to have won in this scenario because the simulatability has been broken. The second case corresponds to the
adversary being able to create a forgery on the MAC. If, and only if, the elements output by the adversary are a forgery, do they meet the requirements of the second equality. Therefore if the second equality holds the adversary has created a forgery and has won the EUF-CMLA game.

We first bound the chance of a collision and then bound the chance of a forgery.

All polynomials initially in lists $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ are of degree one and since the pairing is the only operation that increases the degree of the polynomials, the polynomials can be, at most, degree two. This is because the pairing takes elements from $\mathbb{G}_1$ and $\mathbb{G}_2$ (which are degree one) and the result is put into $\mathbb{G}_3$ and since there is no mapping back from $\mathbb{G}_3$ into other groups, the result follows. Hence, by the Schwartz-Zippel lemma, the probability of two (non-zero) polynomials evaluating to the same value is $\frac{2}{p}$. Since there are, at most, $q$ polynomials, there are $\left(\frac{q}{2}\right) \leq \frac{q^2}{2}$ pairs of polynomials that could collide and thus the probability of any two polynomials colliding is $\frac{q^2}{p}$.

We will now, without loss of generality, focus on leakage in the target group $\mathbb{G}_3$. Any element known to the adversary can be embedded into the corresponding leakage function when required and any group elements in $\mathbb{G}_1, \mathbb{G}_2$ can be transferred into $\mathbb{G}_3$ using a pairing operation with the other group’s generator. Therefore, any element which can be leaked upon from $\mathcal{L}_1$ or $\mathcal{L}_2$ can be leaked on from $\mathcal{L}_3$. Hence, this does not reduce the capability of the adversary.

While only the leakage from $\mathcal{L}_3$ needs to be considered, due to the OCLI assumption, both $\text{Tag}^\mathbb{G}_3$ and $\text{Tag}^\mathbb{G}_3$ will have access to different secret information and, due to the randomness used to update the key, each iteration will have access to different secret information. The lists $\mathcal{L}^\mathbb{G}_3_i$ and $\mathcal{L}^\mathbb{G}_3_i$ will represent all elements which can be calculated by $l^\mathbb{G}_3_i$ and $l^\mathbb{G}_3_i$ respectively.

If the adversary cannot cause a collision of two polynomials, they can only win if they can repeatedly get a (new) tag (or the secret key but this was shown hard by the CKR-UL security proof) into the leakage function to reveal it. If they can only create a forgery once they will not be able to leak the complete tag and thus they are required to get the forgery into multiple leakage sets.

Let $\mathcal{L}^\mathbb{G}_3_i$ be the set of elements that could be computed by the leakage function $l^\mathbb{G}_3_i$. Based on
the leakage set in the theorem we have that:

\[ \mathcal{L}_i^{\bullet} = \{ A \cdot S_i^{\bullet} + B \cdot R_i + C \cdot x_i + D \} \]

Where \( A, B, C \in \mathbb{F}_p \) and \( D \in \mathbb{F}_p \). Without loss of generality, we will assume that \( i \)th tag call maps to \( H_i \).

Let \( \mathcal{L}_i^{\bullet} \) be the set of elements that could be computed by the leakage function \( l_i^{\bullet} \). Therefore:

\[ \mathcal{L}_i^{\bullet} = \{ A \cdot S_i^{\bullet} + B \cdot R_i + C \cdot (x_i + S_i^{\bullet}) + D + e \cdot x_i \cdot H_i \} \]

Where \( e \in \mathbb{F}_p \) and \( S_i^{\bullet} \) denotes \( \sum_{j=0}^{i} R_j \) (corresponding to the definition of \( S_i^{\bullet} \) in the MAC). Without loss of generality, we will assume that \( i \)th tag call maps to \( H_i \).

The only tags that can be contained in \( \mathcal{L}_i^{\bullet} \) are tags which have been received by the adversary from previous queries (and directly embedded into the leakage by the adversary); in this case the tag is already known by the adversary and thus there is no advantage to leaking upon it. Ignoring this trivial case, there are no linear combinations possible that will reveal an unknown tag or the key itself. Similarly, for \( \mathcal{L}_i^{\bullet} \) the only tags that can be included are tags from previous queries and the tag from this query. This is not of any use because the leakage on this tag will be received at the same time the adversary is given the tag and thus no extra information is gained. Tags on previous messages on different randomness can also be learnt but they are of no use to a EUF-CMA adversary, and could be learnt by the adversary without the aid of leakage.

To bound the leakage, we will only consider leakage on elements that involve an unknown group element, since if all elements are completely known by the adversary, the adversary will gain no information from leaking on them. All that is left to show is that no element can be leaked upon more than a bounded number of times (it can be shown to be 3 times):

- \( S_i^{\bullet} \) can be leaked on twice; once in \( \mathcal{L}_{i-1}^{\bullet} \) since \( S_{i-1}^{\bullet} \) is passed in and \( R_i^{\bullet} \) is generated internally (represented by the polynomials \( S_{i-1}^{\bullet} \) and \( R_i \)), and once in \( \mathcal{L}_i^{\bullet} \) since it is passed in.
• $S_i^{\cdot}$ can be leaked on twice; once each in $L_{i-1}^{\cdot}$ and $L_i^{\cdot}$ due to a similar argument as above.

• $R_{i+1}$ can be leaked on twice; once each in $L_i^{\cdot}$ and $L_i^{\cdot}$ since it is generated in $Tag^{\cdot}$ and then passed into $Tag^{\cdot}$ on the $i^{th}$ iteration.

• $S_{i+1}^{\cdot} + X_i$, (a variation of) the intermediated state, can be leaked on 3 times; once in each of $L_i^{\cdot}$, $L_i^{\cdot}$, $L_{i+1}^{\cdot}$. Unlike the previous MAC the intermediate state cannot be calculated in $L_{i+1}^{\cdot}$ because there is no longer access to the randomness $X_i$. It can be calculated in $L_i^{\cdot}$ since the adversary is given $K + X_i$.

Since each element can only be leaked on, at most, three times, the adversary can only learn up to $3 \cdot \lambda$ bits of information per unknown group element. Therefore, the adversary’s advantage can be, at most, $2^{3\lambda}$ times the advantage of playing the standard non-leakage game. This results in the bound given in the theorem statement.

The same argument as for the previous scheme can be used to make successful verifications leak, together with a single failed verification.

### 5.5. A Leakage Resilient Encryption Scheme

This section is based on joint work with Professor Elisabeth Oswald and Dr Martijn Stam [114]. Building on the MAC work, I constructed the encryption scheme and, after discussions with Martijn and Elisabeth, this was formalised into leakage resilient PRFs being added into modes of operation and the authenticated encryption was also considered.

Using similar techniques to those described previously we create a leakage resilient authenticated encryption scheme. Unlike previous schemes in this model which have had security proven directly, we prove security using well known reductions. We first create a leakage resilient PRF and show that certain modes of operation can still be shown to be secure in the face of leakage. Using this, we are able to create a leakage resilient encryption scheme. The technique itself is interesting because most general composition theorems are known to fail in the face of leakage [56].
Before we discuss the construction of the PRF, it is important to first detail why a specifically designed leakage resilient symmetric key encryption scheme is a desirable construction. For example, why construct a symmetric key encryption scheme which only allows leakage on the encryption operation, when a, non-leakage resilient, public key encryption scheme keeping both the public and secret key pair secret could be used. In this scenario, if the public key leaked during encryption, the scheme would still be secure, since security has been shown when the public key is known from the beginning. In the non-leakage setting, the normal argument against this is that symmetric schemes tend to be more efficient than public key schemes, which tend to rely on mathematical group based operations. However, since most leakage resilient cryptography is built using public key techniques (even when it is a symmetric primitive) this is no longer a valid argument (Hazay et al. [89] provide one of a handful of exceptions to this). We will now discuss the advantages to constructing a dedicated symmetric key primitive instead of using an existing public key primitive.

The majority of public key schemes in the literature encrypt a message from a restricted message space, normally a group. However, it is often desirable to be able to encrypt arbitrary messages. It is possible to take a message, split it into multiple parts and encrypt them using a scheme which can only encrypt fixed length messages. For example, ElGamal can only encrypt a single group element but by splitting the message into multiple group elements, arbitrary messages can be encrypted but the ciphertext will be double the length of the original message. The ideal is to have a fixed \textit{stretch constant}, measuring is how much larger the resulting ciphertext is than the original plaintext. The goal is that regardless of how long the plaintext is, the ciphertext is a fixed amount larger than the plaintext.

Probabilistic encryption is an outdated notion when it comes to the “up and coming” field of authenticated encryption. The new preferred notion is nonce (number only once) based, in which the adversary chooses the nonce a plaintext is to be encrypted under, with the restriction that the nonce has not been used to encrypt another message previously (challenge or otherwise). However, no leakage resilient encryption schemes have been given within this security model. There has been a single authenticated encryption scheme secure in the simulatable leakage model [131]. The downside of this scheme is that it requires a leak free component and
uses the older (probabilistic) Left-or-Right security notions for authenticated encryption. It is clear that not all randomness can be controlled by the adversary because if they were allowed to control the randomness that is used to update the secret key (and leak free components were not used) future computation attacks would be possible. Hence, the goal should be to provide leakage resilient schemes which are secure for the nonce based definitions of security and then only introduce (non adversary controlled) randomness to update the key.

In this section we create a leakage resilient authentication encryption scheme with constant stretch. We begin this section with the notion of a leakage resilient PRF and give a construction within the security model. Next we discuss a variety of possible security models for leakage resilient symmetric encryption and discuss which of the notions are the most desirable. We finish the section by using the leakage resilient PRF (along with a leakage resilient MAC) to construct a leakage resilient authenticated encryption scheme within the chosen security model.

### 5.5.1. A Leakage Resilient Pseudo Random Function

In this section we construct a leakage resilient PRF which is secure in the leakage model described at the beginning of the chapter (the leakage resilient security notion for the PRF is given formally below). The PRF is built upon the leakage resilient MAC given in the previous section. However, the proof itself is still of interest because the techniques used to prove the MAC secure, and the techniques used in all previous work, are not applicable here. We explain why this technique does not work in the PRF scenario; it can be seen as a variation of the DKR-UL attack. Before the theorem is given, we introduce a new class of leakage functions which the PRF can be shown to be secure against.

There have been two leakage resilient PRFs within the literature [56, 67]. The first, by Dodis and Pietrzak [56], is leakage resilient providing that the leakage functions are chosen non-adaptively. The leakage resilient PRF of Faust et al. [67] is more efficient than the original scheme by Dodis and Pietrzak, at the expense that both the inputs and the leakage functions must be chosen non-adaptively. For encryption, both the leakage functions and inputs must be able to be chosen adaptively. Therefore, before an encryption scheme can be constructed, we must formalise the required security notion and construct a scheme that meets the requirements.
The reason an adaptive scheme is required is because the goal is to construct an encryption scheme from a PRF in Ciphertext Feed Back (CFB) mode. Since the previous block’s ciphertext is passed as input into the PRF in the next round, to be non-adaptive the adversary would have to predict the PRF output which is a contradiction (if an adversary can predict the output of a PRF for a given input, then they can distinguish it from random).

Security Definitions

Before we can introduce the leakage resilient security model for a PRF, a variation of the PRF security game when leakage is not involved (original definition given in Def. 11) must be introduced. It is clear that the challenge real or random (RR) oracle in the PRF game cannot leak otherwise the adversary can easily win by comparing the leakage to the challenge received. Since this is the only oracle within the game, the game must first be adjusted to include a second oracle to allow leakage. This oracle will be a PRF oracle which allows leakage. A PRF (by nature) is deterministic, and thus domain separation is required to avoid trivial wins. The new game is denoted Explicit PRF security game, as this variation is what would be designed if formally creating the PRF game (before simplification). It is shown to be equivalent to the standard PRF notion.

\[
\text{experiment } \text{Exp}_{E}^{eprf}(A, b): \\
\begin{align*}
& k \xleftarrow{\$} \mathcal{K} \\
& S \leftarrow \{\} \\
& T \leftarrow \{\} \\
& b' \leftarrow A_{\text{RR}(), \text{PRF}()} \\
& \text{Return } b'
\end{align*}
\]

\[
\text{proc } \text{RR}(X): \\
\begin{align*}
& \text{if } X \in S \text{ then} \\
& \quad \text{Return } \$ \\
& \text{end if} \\
& T \leftarrow T \cup \{X\} \\
& \text{if } b = 1 \text{ then} \\
& \quad Y \leftarrow F_{k}(X) \\
& \text{else} \\
& \quad \text{if } W[X] = \bot \text{ then} \\
& \quad \quad W[X] \xleftarrow{\$} \mathcal{Y} \\
& \quad \text{end if} \\
& \quad Y \leftarrow W[X] \\
& \text{end if} \\
& \text{Return } Y
\end{align*}
\]

\[
\text{proc } \text{PRF}(X): \\
\begin{align*}
& \text{if } X \in T \text{ then} \\
& \quad \text{Return } \$ \\
& \text{end if} \\
& S \leftarrow S \cup \{X\} \\
& Y \leftarrow F_{k}(X) \\
& \text{Return } Y
\end{align*}
\]

Figure 5.20.: Explicit PRF Definition
Definition 37 (Explicit PRF (ePRF)). Let $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$ be a family of functions. Then Fig. 5.20 defines the Explicit PRF security game. The advantage of an adversary $A$ winning the game is defined as $\text{Adv}_{eprf}^F(A) = \Pr[\text{Exp}^eprf_F(A, 1) = 1] - \Pr[\text{Exp}^eprf_F(A, 0) = 1]$.

Theorem 10. Given a pseudorandom function family $F : \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, let $A$ be an adversary against the Explicit PRF security of $F$, then there exists an adversary $B$ (of similar complexity to $A$) against the PRF security of $F$ such that:

$$\text{Adv}_{eprf}^F(A) \leq 2 \cdot \text{Adv}_{prf}^F(B).$$

Proof. Let $A$ be an adversary against the Explicit PRF security of $F$. We will use $A$ to construct an adversary $B$ against the PRF security of $F$. Consider an intermediate game $I$ where both the RR oracle and the PRF oracle are random. This gives three games; the real ePRF world (real RR, real PRF), the intermediate game $I$ (random RR, random PRF) and the random ePRF world (random RR, real PRF). If $A$ can win their game with probability $\epsilon$ then it will be possible to distinguish between either the real ePRF world and $I$ or $I$ and the random ePRF world with probability at least $\frac{\epsilon}{2}$. By showing that these games can be simulated by $B$ in such a way that distinguishing between them allows them to win the PRF game completes the proof.

For distinguishing between the real ePRF world and $I$; $B$ can send any queries to either RR or PRF to their RR oracle. Therefore, when the PRF game is in the random world, the game seen by $A$ will be the game $I$, while when the PRF game is in the real world, the adversary $A$ will be playing the Explicit PRF game in the real world. Thus whenever $A$ can distinguish these two games, $B$ can win the PRF game.

For distinguishing between the random ePRF world and $I$; $B$ responds to RR queries randomly and passes on PRF queries to their RR oracle. Since $A$ cannot send the same query to both the RR and the PRF oracle, $B$ is able to do this because it does not matter that both random functions are generated separately. When the PRF game is in the random world, $A$ is playing $I$ while when it is in the real world, $A$ is playing the Explicit PRF random world game. Thus, whenever $A$ can distinguish these two games, $B$ can win the PRF game.

Putting the two results together gives the desired result. \qed

154
**Theorem 11.** Given a pseudorandom function family $F : \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$, let $A$ be an adversary against the PRF security of $F$, then there exists an adversary $B$ (of similar complexity to $A$) against the Explicit PRF security of $F$ such that:

$$\text{Adv}^\text{prf}_F(A) \leq \text{Adv}^\text{eprf}_F(B).$$

**Proof.** It is fairly straightforward to see that any adversary $A$ who can win the PRF game can also win the Explicit PRF game. Since the only difference between the two games is the addition of the PRF oracle and, if it is possible to win without this oracle, then it is possible to win with access to this oracle, by simply not using it. This will allow the adversary to win the Explicit PRF game with at least the same advantage they can win the PRF game, giving the desired result. \[\square\]

With the inclusion of the additional oracle which allows PRF queries we are now in a position to define the leakage resilient notion of a PRF (denoted PRLF). In this security game the adversary can get leakage on any queries made to the PRF oracle. The formal definition of the game is given below.

**Definition 38** (Pseudo Random Leaky Function (\(\mathcal{L}\)-PRLF)). Let $F^* : \mathcal{K}^* \times \mathcal{X} \rightarrow \mathcal{Y}$ be a
family of functions. Then Fig. 5.21 defines the $\mathcal{L}$-PRLF security game. The advantage of an adversary $A$ winning the game is defined as $\text{Adv}_{F^*}^{\mathcal{L} \text{-pRLF}}(A) = \Pr[\text{Exp}_{F^*}^{\mathcal{L} \text{-pRLF}}(A, 1)] - \Pr[\text{Exp}_{F^*}^{\mathcal{L} \text{-pRLF}}(A, 0)] = 1$.

A Leakage Resilient PRF ($F^*$)

\begin{align*}
\text{proc } KG*: \quad & \quad \text{proc } F^*(S^0_i, X): \\
& \quad k \xleftarrow{\$} G_1 \\
& \quad S^0_i \xleftarrow{\$} G_1 \\
& \quad S^0_i \xleftarrow{\$} G_1 \\
& S_0 \leftarrow k \cdot (S^0_i \cdot S^0) \quad \text{Return } (S^0_i, S^0, S^0) \\
& \quad (S^i_{t+1}, O^i_0) \leftarrow F^*(S^i_i, X) \\
& \quad (S^i_{t+1}, O^i_0) \leftarrow F^*(S^i_i, X) \\
& \quad (S^i_{t+1}, Y) \leftarrow F^*(S^i_i, O^i_0, O^i) \\
& \quad \text{Return } Y \\
& \quad R^i_{t+1} \xleftarrow{\$} G_1 \\
& \quad Y^i_{t} \leftarrow e(S^i_i, H(X)) \\
& \quad O^i_{t} \leftarrow (Y^i_{t}, R^i_{t+1}) \\
& \quad S^i_{t+1} \leftarrow S^0_i \cdot R^i_{t+1} \\
& \quad \text{Return } (S^i_{t+1}, O^i) \\
& \quad (Y^i_{t}, R^i_{t+1}) \leftarrow O^i_i \\
& \quad (Y^i_{t}, R^i_{t+1}, X) \leftarrow O^i_i \\
& \quad Y^i \leftarrow e(S^i_i, H(X)) \\
& \quad Y^i \leftarrow Y^i_i \cdot Y^i_{i} \\
& \quad S^i_{t+1} \leftarrow S^i_i \cdot (R^i_{t+1} \cdot R^i_{t+1})^{-1} \\
& \quad \text{Return } (S^i_{t+1}, Y^i) \\
\end{align*}

Figure 5.22.: The PRLF $F^*$

Our PRLF $F^*$ (see Fig. 5.22) is a variation of the leakage resilient MAC given in the previous section. The first step is to show that the MAC $M$ is also a PRF. However, when using the KP key update mechanism $KU$, the scheme is not a PRLF (shown below). This is of interest as it shows that not all schemes which are secure without leakage and compatible with $KU$ (the function to be made leakage resilient is key update splittable) are secure with leakage. Therefore, it will be
impossible to prove a generalised rule for this key update mechanism but schemes will have to be proven secure on a case by case basis. We then show that by splitting the key into three shares, instead of just two, and using the OCLI assumption across three components, the scheme can be shown secure in the face of leakage. While a PRF does not normally have a key generation algorithm (the key is chosen uniformly at random from the key space), for the PRLF a key generation algorithm is required because after the key is chosen uniformly at random it must be shared out. Since \( F^* \) is split into three functions instead of two, the order these functions are called in is also given in Fig. 5.22.

\[
\text{proc } F(k, X):
\]
\[
Y \leftarrow e(k, H(X))
\]
\[
\text{Return } Y
\]

Figure 5.23.: The underlying PRF \( F \)

Before it can be shown that the MAC \( M \)'s tagging algorithm is also a PRF (now denoted \( F \) and recapped in Fig. 5.23 for completeness), a new Diffie-Hellman style problem is required. The Decisional Target Bilinear Diffie-Hellman Problem (DTBDH) is the decisional version of the TBDH problem (Def. 34) which was used to show the security of the MAC \( M \) in the previous section.

**Definition 39** (Decisional Target Bilinear Diffie–Hellman Problem (DTBDH)). Let \( BG = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3, e, p) \) be a set of groups with a pairing between them. The decisional target bilinear Diffie-Hellman problem is then defined as: given \( g_1, g_2, g_3^x, g_3^y, g_3^z \) determine if \( g_3^z = g_3^{xy} \) or \( g_3^z = g_3^r \), where \( x, y, r \) are chosen uniformly at random from \( \mathbb{Z}_p \) and \( z = x \cdot y \) if \( b = 1 \) and \( r \) otherwise for \( b \) chosen uniformly at random from \( \{0, 1\} \). Given an adversary \( A \), their advantage is defined as:

\[
\text{Adv}_{BG}^{\text{dtbdh}}(A) = \Pr[A(g_1, g_2, g_3^x, g_3^y, g_3^{xy}) = 1] - \Pr[A(g_1, g_2, g_3^x, g_3^y, g_3^r) = 1].
\]

If an adversary can win the TBDH problem, then they can be used to win the DTBDH problem. An adversary for the DTBDH problem can be used to solve the DCBDH problem (Def 10). The formal statements are excluded due to the similarity with Theorem 5. Now that the required
assumption is in place, we can show that $F$ is a secure PRF.

**Theorem 12.** Let $H : \{0, 1\}^* \rightarrow \mathbb{G}_2$ be modelled as a random oracle and $A$ be a PRF adversary against $F$ who makes $q_h$ queries to the hash function and $q_r$ queries to the challenge oracle, then there exists an adversary $B$ (of similar computational complexity) against the DTBDH problem such that:

$$\text{Adv}_{F}^{\text{prf}}(A) \leq q_h \cdot q_r \cdot \text{Adv}_{\text{BG}}^{\text{dtbdh}}(B).$$

**Proof.** Let $A$ be an adversary against the query PRF security of $F$ (assume without loss of generality the adversary makes unique queries), then it is possible to construct a series of hybrid games (given in Fig. 5.24). In game $H_i$ the first $i$ queries correspond to real queries, while the remaining queries are responded to with random queries. Game $H_{q_r}$ corresponds to the real world while $H_0$ corresponds to the random world. If $A$ can distinguish between these two
games with probability $\epsilon$ there exists two consecutive games $H_{i-1}, H_i$ that they can distinguish between, with probability at least $\frac{\lambda}{q_h}$. Figure 5.24 shows how an adversary $B$ against the DTBDH assumption can simulate these two games for adversary $A$.

Assume without loss of generality that the value $X$ sent to the PRF on the $i$th query was sent to the random oracle beforehand. If $B$ can guess which query to the random oracle the value $X$ was sent, the reduction will behave as expected. As the oracle will return $g_3^z$, which is either $g_3^{x^y}$ or $g_3^z$ corresponding to the two hybrids $H_i$ or $H_{i-1}$. The probability that this happens is $\frac{1}{q_h}$.

When the adversary $B$ can guess which oracle query the $i$th challenge query is, all queries $j > i$ return random results, while if $j < i$ the RR oracle returns real values (it will not trigger the abort since the $X$ which has the flag set is query $i$). Query $i$ in this scenario returns the DTBDH challenge. It then follows that $z$ is real or random depending if the adversary is playing hybrid $H_i$ or $H_{i-1}$ respectively. Thus, if adversary $A$ can distinguish between these two hybrid games, $B$ can win the DTBDH game. Putting it all together gives the desired result. □

We will now show that security is not possible when $F$ is shared into two shares. The attack is similar to the DKR-UL attack given on $KU$ (Proposition 2). This will justify the need for three shares.

**adversary** $A()$:

1. $X \leftarrow \mathcal{X}$
2. Send $X$ to RR and get back $Y$
3. Set $l \leftarrow e(S_i^\mathcal{H}, X, R_{i+1}^\mathcal{H}) = |e(S_i^\mathcal{H}, H(X))|_\lambda$
4. Set $l \leftarrow e(S_i^\mathcal{H}, Y_i^\mathcal{H}, H(X), R_{i+1}^\mathcal{H}) = |Y \cdot e(S_i^\mathcal{H}, H(X))^{-1}|_\lambda$
5. $X' \leftarrow X$
6. Send $X', l, i$ to PRF and get $Y', \Lambda^\mathcal{H}, \Lambda^\mathcal{H}$
7. Return $(\Lambda^\mathcal{H} = \Lambda^\mathcal{H})$

Figure 5.25: An adversary against $F$ when split into two shares ($L_{prf}$-PRLF)

Figure 5.25 demonstrates an adversary who can win the PRLF game when $F$ is combined with $KU$ using two shares, in the GGM. Therefore $F$ combined with $KU$ is not $L_{prf}$-PRLF secure. Below it is shown that an adversary can win the game with probability $1 - 2^{-\lambda}$. This probability is for a single query attack, using multiple queries an adversary can amplify their advantage.
If $b = 1$ (real world):

$$\Pr[\Lambda^\circ = \Lambda^\circ] = \Pr[l(\mathbf{S}^\circ, \mathbf{R}^\circ, \mathbf{Y}^\circ, \mathbf{H}(X), \mathbf{R}^\circ_{i+1})) = l(\mathbf{S}^\circ, \mathbf{Y}^\circ, \mathbf{H}(X), \mathbf{R}^\circ_{i+1}))]
= \Pr[|e(S^\circ_i, H(X))| = |k \cdot e(S^\circ_i, H(X))^{-1}|]$$

Therefore security is not possible within the two share scenario. We show that when three shares are given, resulting in the scheme in Fig. 5.22, then security is again possible.

Before proving the security of $F^*$ with leakage we must formally define the set of leakage functions $L_3$ which are allowed when the function is split into three. A function $l \in L_3$ is of the following form:

$$l(\langle \mathbf{S}_G^i, \mathbf{S}_G^i, \mathbf{S}_G^i \rangle, \mathbf{X}, \langle \mathbf{R}_G^i, \mathbf{R}_G^i+1 \rangle) = l(\mathbf{S}_G^i, \mathbf{X}, \mathbf{R}_G^i, \mathbf{R}_G^i+1))$$

where $l$, $l^\circ$, $l^\oplus$ are functions which output $\lambda$ bits each and $O^\circ_i$, $O^\oplus_i$ are the output of $F^\circ_i$, $F^\oplus_i$ respectively.

**Theorem 13.** Let $A$ be an adversary against the $L_3$-PRLF security of $F^*$ in the GGM, then there
exists an adversary $B$ (of similar complexity to $A$) against the PRF security of $F$ such that:

$$\text{Adv}_{F^{\mathcal{L}_3}}^{\mathcal{L}_3-\text{prlf}}(A) \leq 2^{4\lambda} \cdot \text{Adv}_{F^*}^{\mathcal{L}_3}(B) + \frac{q^2}{p}$$

where $q$ is the number of queries made to the oracles, $p$ is the size of the group and $\lambda$ is the amount of leakage allowed per share of the PRF, giving $3 \cdot \lambda$ bits leakage per function call.

**Proof.** The proof is given in the GGM and shows that the use of leakage does not allow the adversary to learn any elements that they would be unable to learn if no leakage had been involved. Once this has been shown, it follows that the adversary’s advantage can at most be increased by the number of bits that can be learnt about a single element. By showing that each element is only leaked, at most, four times, the adversary’s advantage can, at most, be increased by $2^{4\lambda}$ over the advantage in the game where no leakage is involved.

As in Theorems 7 and 9, group elements will be represented with polynomials, which will be instantiated at the end of the computation. It will be shown that the chance of an adversary forcing a collision on the polynomials, when instantiated, is small ($\frac{q^2}{p}$).

The oracles used by the GGM are given in Fig. B.1 in App. B.

Let $\mathcal{K}, \{ R_j \}_{j=0}^{q_{HR}}, \{ R_j \}_{j=0}^{q_{HR}}, \{ H_i \}_{j=1}^{q_{HR}}, \{ U_j \}_{j=1}^{2 \cdot q_{RO}}, \{ V_j \}_{j=1}^{2 \cdot q_{RO}}, \{ W_j \}_{j=1}^{2 \cdot q_{RO}}, \{ Y_j \}_{j=0}^{q_{HR}}$ be indeterminants where $q_H$ is the number of hash queries, $q_R$ is the number of calls to the challenge oracle, $q_F$ is the number of calls made to the PRF oracle and $q_O$ is the number of group oracle calls. The indeterminants represent the following: $\mathcal{K}$ is the secret key, $\{ R_j \}_{j=0}^{q_{HR}}, \{ R_j \}_{j=0}^{q_{HR}}$ are the randomness used to update the key, $\{ Y_j \}_{j=0}^{q_{HR}}$ is the output from the challenge oracle, $\{ H_i \}_{j=1}^{q_{HR}}$ represent any hash function queries and $\{ U_j \}_{j=1}^{2 \cdot q_{RO}}, \{ V_j \}_{j=1}^{2 \cdot q_{RO}}, \{ W_j \}_{j=1}^{2 \cdot q_{RO}}$ represent any elements that are guessed in $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_3$ respectively. Let $q = q_H + 8 \cdot q_R + 8 \cdot q_F + 3 \cdot q_O$, the factor 3 arises from the fact the adversary can guess the representation of two elements being passed into a binary operation and learns another from the output, thus adding, at most, 3 elements to the list per oracle call. Calling either the challenge or PRF oracle adds 8 polynomials to the list; the 3 updated shares, the PRF output and an extra element (since the calculation requires two multiplications) and the three intermediate values $Y \odot, Y \circ, Y \oplus$. The lists $\mathcal{L}_1, \mathcal{L}_2, \mathcal{L}_3$ are used to keep track of polynomials
and their representations in $G_1, G_2, G_3$ respectively. They are initialised as follows:

$$L_1 = \{(1, \xi_1^1)\} \cup \{(\gamma_g^\bullet, \xi_i^1)\}^{n}_{j=0} \cup \{(\gamma_g^\circ, \xi_i^1 + q_{g+2})\}^{q_{T}}_{j=0}$$

$$L_2 = \{(1, \xi_1^2)\}$$

$$L_3 = \{(1, \xi_1^3)\}$$

where the $\xi_j^i$ are chosen uniformly at random from $\Xi^i$, such that all polynomials have a unique representation. The sets the representations are drawn from $\Xi^1, \Xi^2, \Xi^3$ (for $G_1, G_2, G_3$ respectively) are all disjoint. All three lists are initially instantiated with the identity. Note that it is not strictly necessary to instantiate the identity in $G_3$ since it can be calculated using the other information provided. We precompute the representations of the randomness used for the key update, the $R_g^\bullet$’s and $R_g^\circ$’s, but since the adversary does not have access to this list of elements, this does not effect the game.

The Adversary $A$ outputs a bit $b'$ and is said to have won if:

1. $f_l^i = f_l^j$ for $l \in \{1, 2, 3\}$ and $i \neq j$

2. $b' = b$

The first case corresponds to the adversary being able to create two polynomials which evaluate to the same value. If this occurs then a single group element has two representations. The adversary is said to have won because the simulation has been broken. The second case corresponds to the adversary being able to distinguish which world they are in. The only way (beyond guessing) that an adversary can distinguish between the two worlds is if they can construct a polynomial $f_i^3$ such that $f_i^3 - k \cdot h = 0$. Where $h$ is the indeterminant corresponding to the hash of an $X$ that has not been sent to the PRF oracle. This corresponds to the adversary winning the PRLF game. We first bound the chance of a collision and then go on to bound the chance of winning the game.

All polynomials originally in $L_1, L_2, L_3$ are of degree one and the only operation that increases the degree is the pairing operation which can only be called on elements in $G_1, G_2$. This
means that degree two polynomials can be in $G_3$ but not the other two lists (since there is no way to get an element of $G_3$ into either of the other two groups). Hence, by the Schwartz-Zippel lemma, the probability of two (non-zero) polynomials evaluating to the same value is $\frac{2}{p}$. Since there are, at most, $q$ polynomials, there are $\left(\frac{q}{2}\right) \leq \frac{q^2}{2p}$ pairs of polynomials that could collide and thus the probability of any two polynomials colliding is $\frac{q^2}{p}$.

Without loss of generality, we will now only look at leakage in the target group $G_3$ since any element from $G_1, G_2$ calculated by the leakage can be transferred over to $G_3$ using a pairing, with the corresponding generator, and any elements known to the adversary can easily be embedded into the leakage by the adversary as required. Since, any element which can be leaked upon from $L_1$ or $L_2$ can be leaked on from $L_3$, by transferring the element to $G_3$ with a pairing, this does not weaken the adversary.

While only the leakage from $L_3$ needs to be considered, due to the OCLI assumption $F^\bigcirc$, $F^\bigcirc$ and $F^\bigcirc$ will have access to different secret information and, due to the randomness used to update the key, each iteration will have access to different secret information. The lists $L_i^\bigcirc$, $L_i^\bigcap$ and $L_i^\bigcirc$ will represent all elements which can be calculated by $l_i^\bigcirc$, $l_i^\bigcap$ and $l_i^\bigcirc$ respectively.

Let $L_i^\bigcirc$ be the set of elements that could be computed by the leakage function $l_i^\bigcirc$. Then, utilising the leakage functions from the theorem statement:

$$L_i^\bigcirc = \{A \cdot S_i^\bigcap + B \cdot R_i^\bigcap + C\}$$

Where $A, B \in \mathbb{F}_p[\{g_i\}_{j=1}^{qH}, \{v_j\}_{j=1}^{2qO}]$ and $C \in \mathbb{F}_p[\{g_i\}_{j=1}^{qH}, \{g_j\}_{j=1}^{qR}, \{u_j\}_{j=1}^{2qO}, \{v_i\}_{i=1}^{2qO}, \{w_i\}_{i=1}^{qO}]$ and $S_i^\bigcirc$ denotes $\sum_{j=0}^i R_j^\bigcirc$ (corresponding to the definition of $S_i^\bigcirc$ in the PRLF).

Let $L_i^\bigcap$ be the set of elements that could be computed by the leakage function $l_i^\bigcap$. Therefore:

$$L_i^\bigcap = \{A \cdot S_i^\bigcap + B \cdot R_i^\bigcap + C\}$$

Where $S_i^\bigcap$ denotes $\sum_{j=0}^i R_j^\bigcap$ (corresponding to the definition of $S_i^\bigcap$ in the PRLF).

Let $L_i^\bigcap$ be the set of elements that could be computed by the leakage function $l_i^\bigcap$. Therefore:
\[ L_i = \{ A \cdot S_i^\triangle + B \cdot R_{i+1} + I \cdot \mathcal{H}_i + C + d \cdot S_i^\circ \cdot \mathcal{H}_i + e \cdot S_i^\circ \cdot \mathcal{H}_i \} \]

Where \( d, e \in \mathbb{F}_p \), \( I \in \mathbb{F}_p[\{ \mathcal{H}_j \}_{j=1}^q, \{ \mathcal{V}_j \}_{j=1}^{2q}] \) and \( S_i^\triangle \) denotes \( \mathcal{K} - \sum_{j=0}^{i}(R_{j+1} - \mathcal{R}_j^\circ) \) (corresponding to the definition of \( S_i^\triangle \) in the PRLF). Without loss of generality we will assume that \( i \)th PRF call maps to \( \mathcal{H}_i \).

The adversary can win if they can leak \( \mathcal{H} \cdot \mathcal{K} \) where \( \mathcal{H} \) corresponds to some unqueried value \( X \). However, there is no such linear combination within the leakage sets that allows this to be possible.

To bound the leakage per element, we will only consider leakage functions which contains at least one unknown group element. Since, while completely known elements can be leaked on multiple times, they do not give the adversary any new information. By showing that each element can only leak a bounded number of times (4 times), the adversary cannot learn any group elements which they would not be able to learn when leakage is not involved and thus the advantage will be increased by at most the number of bits the adversary can learn.

- \( S_i^\triangle \) can be leaked on twice; once in \( L_{i-1}^\triangle \) since \( S_{i-1}^\triangle \) is passed in and \( R_{i+1}^\circ \) is generated internally (represented by the polynomials \( S_{i-1}^\triangle \) and \( \mathcal{R}_i^\circ \)), and once in \( L_i^\triangle \) since it is passed in.
- \( S_i^\circ \) can be leaked on twice; once in \( L_{i-1}^\circ \) since \( S_{i-1}^\circ \) is passed in and \( R_{i+1}^\circ \) is generated internally (represented by the polynomials \( S_{i-1}^\circ \) and \( \mathcal{R}_i^\circ \)), and once in \( L_i^\circ \) since it is passed in.
- \( S_i^\circ \) can be leaked on twice; once each in \( L_{i-1}^\circ \) and \( L_i^\triangle \) due to a similar argument as above.
- \( R_{i+1}^\triangle \) can be leaked on twice; once each in \( L_i^\triangle \) and \( L_i^\circ \) since it is generated in \( F^\triangle \) and then passed into \( F^\circ \) on the \( i \)th iteration.
- \( R_{i+1}^\circ \) can be leaked on twice; once each in \( L_i^\circ \) and \( L_i^\circ \) since it is generated in \( F^\circ \) and then passed into \( F^\circ \) on the \( i \)th iteration.
• $S_i \cdot H_i$, the intermediated state, can be leaked on 4 times, once in each of: $L_{i-1}, L_i, L_{i-1}, L_{i}$ using the argument above for calculating the next share and the fact for input $X_i$, on the $i^{th}$ tag call the intermediate state is generated in $F$ and passed into $F$. To leak on this four times will require querying the same input twice.

• $S_i \circ H_i$, the intermediated state, can be leaked on 4 times, once in each of: $L_{i-1} \circ L_i \circ L_{i-1}, L_i$ using the argument above for calculating the next share and the fact for input $X_i$, on the $i^{th}$ tag call the intermediate state is generated in $F \circ$ and passed into $F$. To leak on this four times will require querying the same input twice.

Since each element can only be leaked on, at most, four times, the adversary can only learn up to $4 \cdot \lambda$ bits of information per unknown group element. Therefore, the adversary’s advantage can be at most $2^{4 \cdot \lambda}$ times the advantage of playing the standard non-leakage game. This results in the bound given in the theorem statement.

It is interesting to note that while, in the three share setting, the adversary can receive more leakage per query ($3 \cdot \lambda$) they cannot receive more leakage per element ($4 \cdot \lambda$). This implies that if similar techniques were used within the MAC the adversary would be able receive more leakage per tagging query with the same level of security, at the expense of efficiency. This leads to the interesting open question of how strong an assumption the OCLI assumption is and how it is affected by the number of shares. We leave this as an interesting open question.

**PRLFs and Leakage Resilient MACs**

It is well known in the literature that any PRF can be converted to a MAC [99]. This has not been considered when leakage is involved. In the case without leakage, a PRF $F$ can become a MAC as follows; the tag of message $M$ is simply $F_k(M)$ and $\sigma$ is verified as a tag of $M$ by checking if $F_k(M) = \sigma$. As hinted at within the MAC section, this relationship does not hold in the face of leakage. By repeatedly sending a message $M$ with some random tag $\sigma$ for verification, an adversary can learn the tag $F_k(M)$ by repeatedly leaking on the verify query. Formally speaking, the ‘standard’ method of converting a PRF into a MAC does not convert a
\( \mathcal{L} \)-PRLF into a \((\mathcal{L}, \mathcal{L})\)-EUF-CMLA MAC. It will however give a \((\mathcal{L}, \{0\})\)-sEUF-CMLA MAC, due to the \( \mathcal{L} \)-PRLF security of the PRLF being used as the tagging algorithm.

### 5.5.2. Leakage Resilient Encryption Notions

One of the most desirable security notions for symmetric encryption today is nonce based authenticated encryption (nAE, Def. 21). Hence, when we include leakage this is the notion that we shall extend. To make the definition remain as strong as possible, when leakage is involved, we want as much to be able to leak as possible. As discussed previously, the challenge oracle is not allowed to leak without the adversary being able to trivially win but both the encryption and decryption oracles should be able to leak. In the nAE setting there are both a challenge encryption and decryption oracle, so the leakage version will have 4 oracles; the two challenge oracles and the two oracles which allow leakage. Since the adversary is allowed to make decryption queries both before and after the challenge queries, they should also be allowed leakage both before and after the challenges. This introduces some difficulties, as seen with DKR-UL (and will be seen again later with certain schemes); it is leakage after a challenge which has allowed certain attacks. The KEM [101] avoided this class of attack by only considering the CCA1 security of their scheme.

#### Definition 40 (Nonce Based Authenticated Leakage Resilient Encryption (\((\mathcal{L}_{\text{enc}}, \mathcal{L}_{\text{dec}})\)-nALE)).

Let \( S\mathcal{E} = (K\mathcal{G}, \mathcal{E}, \mathcal{D}) \) be a symmetric encryption scheme. Then Fig. 5.26 defines the \((\mathcal{L}_{\text{enc}}, \mathcal{L}_{\text{dec}})\)-nALE security game. The advantage of an adversary \( A \) winning the game is defined as

\[
\text{Adv}_{SE}^{(\mathcal{L}_{\text{enc}}, \mathcal{L}_{\text{dec}})\text{-nale}}(A) = \Pr[\text{Exp}_{SE}^{(\mathcal{L}_{\text{enc}}, \mathcal{L}_{\text{dec}})\text{-nale}}(A, 1) = 1] - \Pr[\text{Exp}_{SE}^{(\mathcal{L}_{\text{enc}}, \mathcal{L}_{\text{dec}})\text{-nale}}(A, 0) = 1].
\]

We also give the leakage definition of INDS-CPA (recall Def. 19 without leakage). While this is a weaker notion, and not the end goal, we will require this as a stepping stone to construct our nALE secure scheme. To construct the encryption scheme we will use a similar method to the one which converts any IV based encryption INDS-CPA scheme and PRF into a nAE secure encryption scheme [122].

#### Definition 41 (Indistinguishability From Random Bits Under Chosen Plaintext Leakage Attacks)
experiment $\text{Exp}_{SE}^{(L_{\text{enc}}, L_{\text{dec}})}(A, b)$:

\[
\begin{align*}
k &\xleftarrow{\$} KG \\
S &\leftarrow \{\} \\
T &\leftarrow \{\} \\
U &\leftarrow \{\} \\
V &\leftarrow \{\} \\
b' &\leftarrow A^{\text{RR}(\cdot, \cdot), \text{RP}(\cdot, \cdot), \text{Enc}(\cdot, \cdot, \cdot), \text{Dec}(\cdot, \cdot, \cdot)}() \\
\text{Return } b'
\end{align*}
\]

proc $\text{RR}(N, M)$:

\[
\begin{align*}
\text{if } b = 1 \text{ then } & \\
C &\leftarrow E_k^N(M) \\
\text{else } & \\
C &\leftarrow \mathbb{0} [E_k^N(M)] \\
\text{end if } \\
S &\leftarrow S \cup \{(N, C)\} \\
\text{Return } C
\end{align*}
\]

proc $\text{RP}(N, C)$:

\[
\begin{align*}
\text{if } (N, C) \in U \text{ then } & \\
\text{Return } \perp \\
\text{else if } (N, C) \in T \text{ then } & \\
\text{Return } \perp \\
\text{else if } (N, C) \in S \text{ then } & \\
\text{Return } \perp \\
\text{end if } \\
V &\leftarrow V \cup \{(N, C)\} \\
\text{if } b = 1 \text{ then } & \\
M &\leftarrow D_k^N(C) \\
\text{Return } M \\
\text{else } & \\
\text{Return } \perp
\end{align*}
\]

proc $\text{Enc}(N, M, l)$:

\[
\begin{align*}
\text{if } l \notin L_{\text{enc}} \text{ then } & \\
C &\leftarrow R \mathbb{0} E_k^N(M) \\
\Lambda &\leftarrow l(k, N, M, R) \\
T &\leftarrow T \cup \{(N, C)\} \\
\text{Return } (C, \Lambda)
\end{align*}
\]

proc $\text{Dec}(N, C, l)$:

\[
\begin{align*}
\text{if } l \notin L_{\text{dec}} \text{ then } & \\
\text{Return } \perp \\
\text{end if } \\
\text{if } (N, C) \in V \text{ then } & \\
\text{Return } \perp \\
\text{else if } (N, C) \in S \text{ then } & \\
\text{Return } \perp \\
\text{end if } \\
U &\leftarrow U \cup \{(N, C)\} \\
\text{if } b = 1 \text{ then } & \\
M &\leftarrow R D_k^N(M) \\
\Lambda &\leftarrow l(k, N, C, R) \\
\text{Return } (M, \Lambda)
\end{align*}
\]

Figure 5.26.: nALE experiment

($L$-IND$^*$-CPLA). Let $SE = (KG, E, D)$ be a symmetric encryption scheme. Then Fig. 5.27 defines the $L$-IND$^*$-CPLA security game. The advantage of an adversary $A$ winning the game is defined as

\[
\text{Adv}_{SE}^{L_{\text{ind}}^*\text{-CPLA}}(A) = \Pr[\text{Exp}_{SE}^{L_{\text{ind}}^*\text{-CPLA}}(A, 1) = 1] - \Pr[\text{Exp}_{SE}^{L_{\text{ind}}^*\text{-CPLA}}(A, 0) = 1].
\]

As with the ($L_{\text{tag}}, L_{\text{vrfy}}$)-sEUF-CMLA game, the leakage is described for a general class of leakage functions but it is trivial to convert to the specific OCLI case we are using by substituting in the specific set of leakage functions that correspond to the OCLI case. In these games it is also of note that decryption is a randomised process, where as in the non-leakage setting decryption is defined as a deterministic process. When leakage is involved the randomness will not be used to decrypt the message (in that sense the scheme will still be deterministic) but may be used to update the key. In the nonce based setting this is an even more important distinction, since all
experiment \( \text{Exp}_{\text{SE}}^{L-\text{ind}^C \text{PLA}}(A, b) \):  
\[ k \overset{\$}{\leftarrow} KG \]  
\[ b' \leftarrow A^{\text{RR}(\cdot), \text{Enc}(\cdot)}() \]  
Return \( b' \)

\[ \text{proc RR}(M): \]  
\[ \text{if } b = 1 \text{ then} \]  
\[ C \leftarrow E_k(M) \]  
\[ \text{else} \]  
\[ C \overset{\$}{\leftarrow} \{0, 1\}^{|E_k(M)|} \]  
\[ \text{end if} \]  
\[ \text{Return } C \]

\[ \text{proc Enc}(M, l): \]  
\[ \text{if } l \not\in L \text{ then} \]  
\[ \text{Return } \sharp \]  
\[ \text{end if} \]  
\[ C \overset{R}{\leftarrow} E_k(M) \]  
\[ \Lambda \leftarrow l(k, M, R) \]  
\[ \text{Return } (C, \Lambda) \]

**Figure 5.27:** \( L \)-IND\$-CPLA experiment

the randomness (on the output) should be supplied by the nonce, the internal randomness should only be used to update the key.

The IND\$-CPLA notion captures security of leakage on encryption only and makes no security claims against decryption leakage. The goal is to use an IND\$-CPLA scheme as a building block to construct a nALE secure scheme. However, the goal is that the decryption of the resulting nALE scheme is also allowed to leak. Therefore, the underlying scheme must also be allowed to leak on decryption. IND\$-CPLA does not provide this. Consider an IND\$-CPA secure scheme where encryption cannot leak but decryption outputs the key. This scheme will be IND\$-CPLA secure but will not be secure under decryption. We present a stronger notion called Augmented Indistinguishability From Random Bits Under Chosen Plaintext Leakage Attacks (aIND\$-CPLA). This notion augments the IND\$-CPLA notion with a decryption oracle which only accepts ciphertexts which were output from the encryption oracle. Therefore, the only information an adversary will learn from this oracle is new leakage, since they must already know the corresponding plaintext. This notion captures that leakage on decryption of certain ciphertexts is not too detrimental to the security of the scheme.

**Definition 42** (Augmented Indistinguishability From Random Bits Under Chosen Plaintext Leakage Attacks (aIND\$-CPLA)). Let \( \text{SE} = (KG, E, D) \) be a symmetric encryption scheme. Then Fig. 5.28 defines the \( (L_e, L_d) \)-aIND\$-CPLA security game. The advantage of an adversary \( A \) winning the game is defined as  
\[ \text{Adv}_{SE}^{(L_e, L_d) \text{-aIND}^\$ \text{CPLA}}(A) = \Pr[\text{Exp}_{SE}^{(L_e, L_d) \text{-aIND}^\$ \text{CPLA}}(A, 1) = 1] - \Pr[\text{Exp}_{SE}^{(L_e, L_d) \text{-aIND}^\$ \text{CPLA}}(A, 0) = 1]. \]

The notion can be considered to be the case where an adversary can make encryptions and measure the leakage but can only measure the leakage on the decryption of honesty generated
Figure 5.28.: aIND$^s$-CPLA experiment

ciphertexts. This may correspond to scenarios where the ciphertext is checked for authenticity (using a MAC for example) before being decrypted. This is the scenario we will be using. In this case the adversary will only be able to measure leakage of ciphertexts they received from encryption. aIND$^s$-CPLA is not a desirable goal in its own right but is useful as a stepping stone to schemes secure against stronger security notions.

It is possible to change the games above between nonce based, probabilistic and IV based. Due to the similarities of these notions with the ones given above we do not give the experiments here.

**Alternative Definitions**

The nALE definition is incredibly strong, in that the adversary has a lot of power. This can make the notion hard to achieve. We now give some alternative definitions, that are slightly weaker, in which the adversary is restricted to when they are allowed leakage. In the leakage resilient MAC section, there was the interesting discussion as to what function would be allowed to leak
Within the security model. This was the first time this type of consideration had been made and in this section we build upon it in the setting of leakage resilient encryption.

In the non-leakage model, other than AE, there are three standard definitions for encryption; IND-CPA (Def. 14), IND-CCA1 (Def. 15) and IND-CCA2 (Def. 16). For each of these three games we need to decide what is allowed to leak. No leakage results in the original games and therefore either encryption or decryption (or both) must leak. Due to the nature of some of the games, not allowing decryption queries, we do not consider the case where decryption is allowed to leak but encryption is not (although it can be captured by an empty encryption leakage set). The Release of Unverified Plaintexts (RUP) notion of AE security by Andreeva et al. [7], where decryption gives the adversary unverified plaintexts, can be seen as a notion which allows leakage on decryption but not encryption. However, they are focusing on the more specific case of AE where there is always a decryption oracle, while we consider more general cases where the adversary may not be access to a decryption oracle. Matching the corresponding security notions, leads to having encryption leak and three variations of decryption leakage; no decryption leakage (L0), decryption leakage only before a challenge has been requested (L1) and having decryption always allowed to leak (L2).

While the sets of leakage functions capture what an adversary is allowed to leak, the leakage functions are not aware of when within the game they are being called and therefore cannot leak differently depending on the moment in time called. This means that the class of leakage functions themselves are unable to capture the L0, L1 or L2 aspect of the game. Hence, this is an interesting distinction between the resulting security notions.

This results in nine different security games for encryption IND-CPA-LX, IND-CCA-Y-LX for $X \in \{0, 1, 2\}, Y \in \{1, 2\}$. If the leakage function is more powerful than the ability of the adversary to decrypt messages (such as the IND-CPA-L1 scenario where the adversary can have no decryption queries but can leak on decryption before the challenge), then the decryption oracle will only return the leakage and not the corresponding plaintext.

For certain classes of leakage functions it is possible to show that if the decryption leakage is more powerful than the adversary’s ability to decrypt then they can use the leakage function to decrypt the message instead. Consider the illustrative example in which the adversary can
adaptively choose leakage functions that leak a single bit of plaintext. In this case, by repeatedly sending the same ciphertext to the decryption oracle, the adversary can completely learn the plaintext by leaking on the decrypted value one bit at a time. In this scenario, certain security notions are equivalent to others, resulting in six unique security definitions. For example, IND-CCA1-L2 will be equivalent to IND-CCA2-L2 under these particular classes of leakage.

The notion $\mathcal{L}$-IND$^-$CPLA given above would be denoted $\mathcal{L}$-IND$^-$CPLA-L0 under this new naming convention.

Where attacks are given in the remainder of the chapter, the full power of L2 leakage is required, since leakage is required after a challenge has been received. Therefore, while IND-CCA1 is not normally considered an interesting or desirable notion when leakage is not involved, it may be a more interesting notion when leakage is involved. This may allow schemes to be created which are secure against certain types of leakage without being overly complex to protect against attacks which are more model relics than viable practical attacks. In practice, the IND-CCA1-L1 attack relates to having brief access to a device to measure side channel leakage and then not having access to the device from when the challenge is seen onwards. The leakage resilient KEM [101] is given within this model.

5.5.3. A Leakage Resilient Encryption Scheme

For the remainder of this section we create a leakage resilient encryption scheme that is secure in the nALE model. To do this we show that a leakage resilient PRF in certain modes of operation gives an encryption scheme which is IND$^-$CPLA secure, where the leakage for each encryption behaves as follows. Since a message is multiple blocks long, the encryption scheme will work by making a single call to the leakage resilient PRF for each message block that is encrypted and thus the adversary can submit a leakage functions per message block (for a concrete instantiation using the PRLF given this will be a triple of leakage functions due to the OCLI assumption). All leakage functions to the PRF, corresponding to a single encryption, must be submitted at the same time (they can not depend on the output of the other functions for the same encryption query). We feel that this more accurately represents how a device behaves in practice, since the longer the message is to be encrypted, the more leakage that the adversary can learn. This class
of leakage functions will be defined formally below. Once we have an aIND$\text{-CPLA}$ secure scheme we show how it can be combined with other leakage resilient components to give a nALE secure scheme.

**Leakage Resilient Modes of Operation**

The mode of operation we have chosen to create our leakage resilient encryption scheme is CFB mode. For completeness, CFB mode is shown in Fig. 5.29. The operation $*$ used in CFB mode will vary depending on the PRLF being used.

The proof that CFB mode provides a IND$\text{-CPLA}$ mode of operation when instantiated with a PRLF follows extremely closely to the proof that CFB mode is a secure mode of operation [5]. The class of leakage functions $L_{\text{CFB}} = \{l : \mathcal{K} \times \mathcal{IV} \times (\mathcal{M} \times \mathcal{R})^+ \rightarrow \{0, 1\}^\ast\}$ that CFB mode can be shown to be secure against contains leakage functions of the following form:

$$l_{\text{CFB}}(k, IV, M, R) = \{l_i(k, C_i, R_i)\}_{i=0}^{n-1}$$

where $E_k(IV) = C$, $M$ is an $n$ block message, $C_i$ is the $i$th ciphertext block ($C_0 = IV$), $R_i$ is the randomness required by the $i$th PRLF call and $l_i \in L_{\text{PRLF}}$.

Due to the nature of the proof, the PRLF must be secure against adaptive leakage queries; the PRLF given previously meets this requirement under the OCLI assumption. Since no computation occurs outside of the PRLFs on unknown values these leakage functions capture leakage on all data. Due to representing leakage in this form, the longer the message, the more leakage
received by the adversary, which closely models what happens in reality.

**Theorem 14.** Let \( \mathcal{SE} = (\mathcal{KG}, \mathcal{E}, \mathcal{D}) \) be the symmetric encryption scheme formed when the PRLF \( F^* : \mathcal{X} \rightarrow \mathcal{Y} \) is put into CFB mode. Let \( A \) be an adversary against the \( L_{CFB}^{\text{IND$_*$-CPLA}} \) security of \( \mathcal{SE} \). Then there exists an adversary \( B \) (of similar complexity to \( A \)) against the \( L_{PRLF}^{\text{PRLF}} \) security of \( F^* \) such that:

\[
\text{Adv}_{\mathcal{SE}}^{L_{CFB} - \text{ind$cpla}}(A) \leq \text{Adv}_{\mathcal{SE}}^{L_{PRLF} - \text{prlf}}(B) + \frac{q_e^2 \cdot s^2}{|\mathcal{X}|}
\]

where \( q_e \) is the number of encryption calls (both to challenge and encryption oracles) and \( s \) is the length of the longest message encrypted.

**Proof.** The proof of security follows as a series of game hops between games \( G_1 \) and \( G_3 \). It can be seen that game \( G_1 \) (Fig. 5.30) is exactly the real world of the IND$\_*$-CPLA game when encryption is instantiated with CFB mode. The game \( G_3 \) (Fig. 5.32) is the random world of IND$\_*$-CPLA. The randomness is generated in blocks for clarity but this does not change the result compared to generating the entire ciphertext at once. Bounding the advantage of an adversary between \( G_1 \) and \( G_3 \) also bounds the advantage of the adversary winning the IND$\_*$-CPLA game. To help bound this term we introduce an intermediate game \( G_2 \) (Fig. 5.31) in which the calls to the PRF have been replaced with calls to a truly random function (with memory).
experiment $\text{Exp}_{\mathcal{G}_2}^G(A)$:
\begin{align*}
&k \xleftarrow{\$} K \\
&b' \leftarrow A^{\text{RR}(\cdot),\text{Enc}(\cdot)}() \\
\text{Return } b'
\end{align*}

proc $\text{RR}(M)$:
\begin{align*}
&M_1 \ldots M_n \leftarrow M \\
&C_0 \xleftarrow{\$} \mathcal{Y} \\
&\text{for } i = 1 \text{ to } n \text{ do} \\
&\quad \text{if } F[C_{i-1}] = \perp \text{ then} \\
&\quad \quad F[C_{i-1}] \xleftarrow{\$} \mathcal{Y} \\
&\quad \text{end if} \\
&\quad C_i \leftarrow F[C_{i-1}] \star M_i \\
&\text{end for} \\
&C \leftarrow C_0 \ldots C_n \\
\text{Return } C
\end{align*}

proc $\text{Enc}(M,l)$:
\begin{align*}
&M_1 \ldots M_n \leftarrow M \\
&l_1 \ldots l_n \leftarrow l \\
&C_0 \xleftarrow{\$} \mathcal{Y} \\
&\text{for } i = 1 \text{ to } n \text{ do} \\
&\quad F_i \leftarrow F^*_k(C_{i-1}) \\
&\quad C_i \leftarrow F_i \star M_i \\
&\quad \Lambda_i \leftarrow l_i(k,C_{i-1},R_i) \\
&\text{end for} \\
&C \leftarrow C_0 \ldots C_n \\
&\Lambda \leftarrow \Lambda_1 \ldots \Lambda_n \\
\text{Return } (C,\Lambda)
\end{align*}

Figure 5.31.: The game $G_2$, used to prove IND$^*$-CPLA security of CFB mode

To complete the proof, the advantage of distinguishing $G_1$ from $G_2$ and $G_2$ from $G_3$ needs to be bounded. An adversary will be able to distinguish $G_2$ and $G_3$ only when they can cause a collision in the random function calls, which happens with probability $\frac{q^2}{|X|}$ (by the birthday bound), where $q$ is the number of function calls. If an adversary can cause a collision, it means the same value will be input into the random function twice, but since, in game $G_3$ the random function has no memory, it will return two different values. This will allow the adversary to distinguish between the two games. An adversary who can distinguish between if they are playing game $G_1$ or game $G_2$ can be used to distinguish between the real and random world of a PRLF: the reduction is given in Fig. 5.33.

Figure 5.32.: The game $G_3$, used to prove IND$^*$-CPLA security of CFB mode
Putting it all together and using the fact that $q$ the number of PRLF calls is at most $q_e \cdot s$ where $q_e$ is the number of encryption queries and $s$ is the number of blocks in the longest message gives the desired result.

\[
\text{adversary } B: \quad b' \leftarrow A^{RR(\cdot, \cdot), Enc(\cdot, \cdot)(\cdot)}
\]

Return $b'$

\[
\text{simulator } RR(M): \quad M_1 \ldots M_n \leftarrow M
\]

$C_0 \leftarrow Y$

\[
\text{for } i = 1 \text{ to } n \text{ do}
\]

$F_i \leftarrow RR(C_{i-1})$

$C_i \leftarrow F_i \star M_i$

\[
\text{end for}
\]

$C \leftarrow C_0 \ldots C_n$

Return $C$

\[
\text{simulator } Enc(M, l): \quad M_1 \ldots M_n \leftarrow M
\]

$l_1 \ldots l_n \leftarrow l$

$C_0 \leftarrow Y$

\[
\text{for } i = 1 \text{ to } n \text{ do}
\]

$(F_i, \Lambda_i) \leftarrow \text{PRF}(C_{i-1}, l_i)$

$C_i \leftarrow F_i \star M_i$

\[
\text{end for}
\]

$C \leftarrow C_0 \ldots C_n$

$\Lambda \leftarrow \Lambda_1 \ldots \Lambda_n$

Return $(C, \Lambda)$

Figure 5.33.: A PRLF adversary $B$ constructed using a distinguisher $A$ for $G_1, G_2$

This proof assumes that the PRLF outputs random bits. If the PRLF outputs elements from a different set (such as group elements) then the resulting encryption scheme will be indistinguishable from a sequence of elements from this set. Our PRLF above outputs elliptic curve points and thus the output from CFB will be indistinguishable from random elliptic curve group elements. It has been shown how to encode elliptic curve points such that they are indistinguishable from random bits [10, 26, 161]. Therefore, applying one of these encodings to our PRLF will give a scheme which is indistinguishable from random bits.

It is reasonably straightforward to take the proof given and change it to show $L_{\text{CFB}}$-IND-CPLA security instead of the indistinguishability from random bits security given. This may be desirable as people tend to find the more traditional, Left-or-Right security more intuitive to work with. It may be possible to show that the relation given between IND$\$-CPA and IND-CPA in Sect. 2.3, also holds in the face of leakage. For other modes of operation, such as Output Feed Back (OFB) and Counter (CTR) mode, the proof of security is extremely similar. Interestingly, while the generic proof goes through for modes such as Ciphertext Block Chaining (CBC) mode, there is currently no block cipher which is secure in the continuous leakage model: we leave this as future work.
In CFB mode, encryption and decryption are extremely similar in terms of operations performed. In encryption the ciphertext (starting with $C_0$) is repeatedly encrypted and the output of this is exclusive-ored with the message block to create the next ciphertext block. In decryption the ciphertext (starting with $C_0$) is repeatedly encrypted and the output of this is exclusive-ored with the next ciphertext block to retrieve the corresponding plaintext block. Due to their similarities, if encryption and decryption are implemented on the same device they will leak in an extremely similar fashion. In this scenario it makes sense to have the sets of leakage functions for encryption and decryption to be the same. In our particular case, it is reasonably straightforward to see that if both encryption and decryption are allowed to leak from $L_{CFB}$, then the adversary cannot do better in the aIND$^+$-CPLA game than when just given encryption access (since decryption can just been seen as a series of calls to $F^*$ which is secure by the assumption that $F^*$ is a PRLF). Therefore, the encryption scheme from, CFB mode is $(L_{CFB}, L_{CFB})$-aIND$^+$-CPLA secure, for our particular choice of leakage functions.

**Generic Composition in the Face of Leakage**

The Competition for Authenticated Encryption: Security, Applicability, and Robustness (CAE-SAR) [25] was proposed to help standardise a new authenticated encryption scheme. This lead to a variety of schemes with a whole host of different security properties [1,8,90]. More recently the authenticated encryption community have considered stronger notions of security which can be seen as protecting against specific forms of leakage. Andreeva et al. [7] consider RUP security where the decryption oracle will give the adversary the plaintext regardless of if verification succeeds. This can be seen as leaking the plaintext in the situations where verifications fails, with no leakage on a successful verification. They go on to show a scheme which is secure within this model. Robust Authenticated Encryption (RAE) [90] makes the stretch a user controlled parameter. This allows the adversary to choose how much longer than the plaintext each ciphertext will be. Since decryption accepts any length ciphertext, it cannot be possible to trivially change the length of a valid ciphertext. One option is to add redundancy to the message, padding it to the required length before it is encrypted. It is shown that this method is RAE secure and also achieves security when plaintexts are released where the redundancy is incorrect. This revealing
of plaintexts with incorrect redundancy can be seen as a form of leakage. An earlier piece of research which fits into the same “class of leakage” is distinguishable decryption. In this model the decryption oracle can output different types of error symbol. For example, in RSA OAEP, if different errors are given for authentication errors and padding errors, an adversary can decrypt a message without knowing the secret key [112]. Boldyreva et al. [31] construct security models which allow the adversary to distinguish various decryption errors. Barwell et al. [15] reconcile the RUP, RAE and distinguishable decryption goals, showing equivalences of the notions under minor restrictions.

The other topic which has been considered for authenticated encryption, long before the CAESAR competition, is generic composibility. That is to say, constructing authenticated encryption schemes from components with certain security properties, regardless of how they work. The original composition paper [21] showed the various ways to combine a probabilistic IND-CPA secure encryption scheme with a sEUF-CMA secure MAC to produce a probabilistic IND-CCA2 secure encryption scheme. In this paper they conclude that out of the three options; Encrypt-then-MAC, MAC-then-Encrypt and Encrypt-and-MAC, the only one which is generically secure is Encrypt-then-MAC. Namprempre et al. [122] revisit generic composition of encryption schemes with MACs and/or PRFs to construct encryption schemes. They consider combining an IV based IND$^\$-CPA secure encryption scheme and a PRF to produce a nAE secure encryption scheme.

In the most recent composition paper [122] it is shown there are 160 ways to combine a PRF, which can naturally be used as a MAC, with an IND-CPA secure IV based encryption scheme. Of these 8 are generically secure, 1 is transitional (it is secure but with an inferior bound), 3 have yet to be proven either secure or insecure and the remaining 148 schemes either do not make sense or have been shown to be insecure. From this point forward we will focus on the 8 ‘favoured’ schemes A1-A8 where A4 is known as Synthetic IV (SIV) in the literature. We will now discuss the security of these schemes when leakage is involved, under the nALE security notion. While it is possible to discuss their security properties under some of the other notions given in this work, since the other notions are weaker, we do not consider them any further here.

For certain classes of leakage function (including the one considered here), it is unfortunately
not possible to achieve nALE security (or the weaker notion of IND-CCA-L2). These schemes are A1, A2, A3, A4 (SIV), A7 and A8. The issue with these schemes is that they all perform a variation of MAC-then-Encrypt or MAC-and-Encrypt. In these cases the tag is given on the message and not the ciphertext and this allows the following attack to be performed. Since the tag is on the message, the decryption algorithm must first decrypt the message before the validity of the message can be checked. Thus, when an adversary is given a challenge ciphertext they can change the tag to something different and repeatedly send this to decrypt. While decrypt will fail and not return anything (otherwise the adversary has broken the sEUF-CMA security of the MAC) they are able to leak on the message after it has been decrypted which will allow them to win the game. These schemes will also be insecure for the RUP notion of security, since given a challenge the adversary can change the tag and while verify will fail the adversary is given the unverified plaintext allowing them to see if it was the message sent to the challenge oracle.

The initial belief was that Encrypt-then-MAC was the most desirable method of generic construction [21]. However, recent work [122] shows that this is only the case when an outdated notion of encryption is used, and when a newer definition is used, it is possible for certain Encrypt-and-MAC and MAC-then-Encrypt generic compositions to be as equally desirable as the Encrypt-then-MAC compositions. The above attack shows that when leakage is involved MAC-then-Encrypt is once again the most desirable composition method, as the other methodologies \textit{have} to allow leakage on the underlying message before validity is checked.

![Diagram of A5 method of generic composition]

Figure 5.34.: A5

From this point forward, we will focus on the A5 method of generic composition, which is
recapped in Fig. 5.34 for completeness (not to be confused with the A5 family of stream ciphers used within the GSM cellular telephone standard [36]). The nonce $N$ is used to generate the $IV$ for the encryption scheme, which encrypts the message $M$. The nonce, ciphertext and associated data $A$ are put through the PRF to generate the tag. Associated data is information which is authenticated but not encrypted. For clarity, we have excluded it from our definitions (both nAE and nALE), and will exclude it from the proof that follows. However, it is trivial to include it in both the definitions and proof, if required. The reason for showing the security of A5 is that it is trivial to extend the proof to show security for A6 in which the associated data is also passed into the PRF to generate the IV. Since the security of A5 is known [122], “all” that remains is to show its security when leakage is also considered.

The generic composition A5 can be seen as an Encrypt-then-MAC scheme. However, it will not be secure if the scheme still decrypts the message after a failed tag verification. If the scheme decrypts the message before returning the fail error for the MAC then the scheme will be insecure for the same reasons the MAC-then-Encrypt schemes were. Thus from this point forward, if a verification query fails we assume that decryption terminates without calculating the plaintext and therefore only giving the adversary leakage on the tag. This will be made formal when the leakage sets for A5 are defined.

As shown previously, a $\mathcal{L}$-PRLF does not necessarily provide a $(\mathcal{L}, \mathcal{L})$-sEUF-CMLA secure MAC. Thus, while in the non-leakage resilient version only a PRF and encryption scheme are required (since the PRF provides a MAC), in the leakage setting a PRLF, a leakage resilient MAC (where the tag function is also a PRLF) and leakage resilient encryption scheme are required. While we provide a leakage resilient MAC earlier in the chapter, we are unfortunately not able to provide a $(\mathcal{L}_{\text{tag}}, \mathcal{L}_{\text{vrfy}})$-sEUF-CMLA MAC, which is required for the generic composition, and unfortunately no $(\mathcal{L}_{\text{tag}}, \mathcal{L}_{\text{vrfy}})$-sEUF-CMLA is known within the literature either (for $\mathcal{L}_{\text{vrfy}}$ not equal to the constant function). However, for our construction (until such a scheme is known) a leakage resilient signature scheme can be used (e.g. [75]), where the secret key is the signature scheme’s $(pk, sk)$ pair. Verification is trivially secure under leakage since, even if the adversary learns $pk$, the scheme will still be secure because, in the non-leakage signature notion, the adversary initially knows this value. The tagging oracle would still need to be shown to be a
We now show the security of A5 under leakage. Based on the previous discussions, a $L_{prf}$-PRLF to create the IV from the nonce, an $(L_{enc}, L_{dec})$-aIND$^*$-CPLA secure encryption scheme and a $(L_{tag}, L_{vrfy})$-sEUF-CMLA leakage resilient MAC are required. The leakage we are considering is as follows; leakage $l_e \in L_e$ for encryption is of the form $l_e : \mathcal{K} \times \mathcal{N} \times \mathcal{M} \times \mathcal{R} \to \{0, 1\}^*$ and leakage $l_d \in L_d$ for decryption is of the form $l_d : \mathcal{K} \times \mathcal{N} \times \mathcal{C} \times \mathcal{R} \to \{0, 1\}^*$. These functions are then defined as:

$$l_e((k_1, k_2, k_3), N, M, (R_{prf}, R_{enc}, R_{tag})) = l_{prf}(k_2, N, R_{prf}) || l_{enc}(k_1, IV, M, R_{enc}) || l_{tag}(k_3, C || N, R_{tag})$$

$$l_d((k_1, k_2, k_3), N, (C, \sigma), (R_{vrfy}, R_{prf}, R_{dec})) = \begin{cases} l_{vrfy}(k_3, C || N, \sigma, R_{vrfy}) & \text{if } v = 0 \\ l_{vrfy}(k_3, C || N, \sigma, R_{vrfy}) || l_{prf}(k_2, N, R_{prf}) || l_{dec}(k_1, IV, C, R_{dec}) & \text{otherwise} \end{cases}$$

For $l_{prf} \in L_{prf}, l_{enc} \in L_{enc}, l_{dec} \in L_{dec}, l_{tag} \in L_{tag}, l_{vrfy} \in L_{vrfy}$ and $IV = F_{k_2}(N), C = \mathcal{E}_{k_3}^{IV}(M), v = \text{VRFY}(k_3, (N, C), \sigma)$ using randomness $R_{prf}, R_{enc}, R_{mac}$ respectively.

The encryption leaks independently on the results of the IV generation from the nonce, the encryption of the message under the IV and the tagging of the ciphertext. Since they each use an independent key, and are used at different moments in time, they should leak independently in practice. Similarly the randomness used by each individual component may only be leaked on at the time it is used. If decryption fails the verify, only verify will leak because decryption will terminate at this stage, therefore not allowing leakage on the IV generation or the decryption. However, if verify is successful, the IV generation and the decryption of the ciphertext will be also allowed to leak. Again, due to the separation in time these secret values and randomness will leak independently. We again emphasis that this condition on the MAC verification is security critical. Any operation for the PRF or encryption that began before the completion of verify would invalidate the proof (using the attack described previously).

Due to the nature of the leakage reduction, all schemes are required to be secure against adaptive leakage. The way that data is passed within the scheme means that representing the
leakage functions in this manner captures all leakage. Since no computation will occur outside of the PRF, encryption or MAC but data is simply passed between the functions, all the leakage is captured.

**Proof Intuition** While the proof itself is fairly involved, it is straightforward to give a high level intuition as to why it holds. For both encryption and decryption (see Fig. 5.34) there are no internal wires. While the IV appears to be internal, security of the underlying encryption scheme holds when the adversary is given the IV. Therefore all wires going into any leakage resilient component are either inputs to the whole scheme or outputs from the entire scheme. Leakage composability proofs tend to fail when there is an element output by one component and passed directly into another component without being output, but only ever being used internally. This is because leakage on these could allow the adversary to learn elements they are not supposed to know thus breaking the scheme, while not breaking the security of the underlying component where the element is assumed to be output. The other issue could be that now unknown elements are being passed into a leakage function and thus the reduction cannot be made as it is not clear what this element is when leakage is requested upon it. However, none of this is an issue for A5 and thus it is possible for the reduction to go through.

We now give the theorem statement and formalise this proof.

**Theorem 15.** Let $SE^*$ be a $(L_{enc}, L_{dec})$-aIND$^S$-CPLA secure symmetric encryption scheme, $M^*$ be a $(L_{tag}, L_{vrfy})$-sEUF-CMLA secure MAC (which is also a $L_{tag}$-PRLF) and let $F^*$ be a $L_{prf}$-PRLF. If $A$ is the encryption scheme resulting in using $(SE^*, M^*, F^*)$ in mode A5 then $A$ is $(L_e, L_d)$-nALE secure, such that:

$$Adv_A^{(L_e, L_d)\text{-nale}}(A) \leq Adv_{M^*_{\text{seufcmla}}}^{(L_{tag}, L_{vrfy})-\text{sEUF-CMLA}}(B) + Adv_{SE^*_{\text{aind$^S$-CPLA}}}(C) + Adv_{F^*_{\text{PRLF}}}(D) + Adv_{M^*_{\text{PRLF}}}(E).$$

**Proof.** The game $G_a$ (Fig. 5.35) corresponds to the real nALE world where the challenge oracles RR and RP return real ciphertexts and perform valid decryptions respectively. The game $G_e$ (Fig. 5.39) corresponds to the ideal nALE world where the challenge oracles RR and RP return
random bitstrings for ciphertexts and always return fail on decryption. Bounding an adversary’s advantage of distinguishing these two games bounds the advantage the adversary can win the nALE game. To do this we introduce three intermediate games \((G_b, G_c, G_d)\).

The difference between the games \(G_a\) and \(G_b\) (Fig. 5.36) is that \(G_b\) always returns fail from RR instead of correctly decrypting a message. The only way for an adversary to distinguish between these two worlds is to pass a valid ciphertext to RP which has not come from Enc or RR (as trivial wins have been excluded). Therefore, the only way for an adversary to distinguish the two games is via creating a forgery on the MAC \(M^*\).

The game \(G_c\) (Fig. 5.37) is a variation of \(G_b\) where the PRLF has been replaced with a truly random function in RR. If an adversary can distinguish between \(G_b\) and \(G_c\), they can be used to construct an adversary against the PRLF \(F\).

Since the IV for encryption is now truly random (as assumed in the IV schemes proof of security), the encryption scheme can be switched for a truly random function. Game \(G_d\) switches the ciphertext in RR to a random string. Any adversary who can distinguish between \(G_c\) and \(G_d\) (Fig. 5.38) can be used to win the aIND$-CPLA security game of the underlying encryption scheme.\n
The final game \(G_e\) switches the tag to be a random string. If an adversary can distinguish between \(G_d\) and \(G_e\), they can be used to construct an adversary against the PRLF provided by the tagging function of \(M^*\).

Putting this all together gives the desired result. \(\square\)

Shrimpton and Terashima [155] define a (weaker) authenticated encryption notion where the ‘recovery information’ does not need to be random, only the ciphertext. Therefore, if for the given purpose the tag does not have to be indistinguishable from random, the tagging algorithm would not have to also be a PRLF.
experiment $\text{Exp}_{\mathcal{A}}^{\text{Ga}}(A)$:

- $S \leftarrow \{\}$
- $T \leftarrow \{\}$
- $\mathcal{U} \leftarrow \{\}$
- $\mathcal{V} \leftarrow \{\}$
- $k_1 \xleftarrow{} \mathcal{KG}$
- $k_2 \xleftarrow{} \mathcal{KG}$
- $k_3 \xleftarrow{} \mathcal{KG}$
- $b' \leftarrow A^{\text{RR}(\cdot), \text{RP}(\cdot), \text{Enc}(\cdot), \text{Dec}(\cdot)}$
- Return $b'$

proc $\text{RR}(N, M)$:

- $IV \leftarrow F_{k_2}^*(N)$
- $C \leftarrow E_{k_1}^IV(M)$
- $\sigma \leftarrow \text{Tag}(k_3, C || N)$
- $S \leftarrow S \cup \{(N, C, \sigma)\}$
- Return $(C, \sigma)$

proc $\text{RP}(N, C, \sigma)$:

- if $(N, C, \sigma) \in \mathcal{U}$ then
  - Return ⊥
- else if $(N, C, \sigma) \in \mathcal{T}$ then
  - Return ⊥
- else if $(N, C, \sigma) \in S$ then
  - Return ⊥
- else if $(N, C, \sigma) \in \mathcal{V}$ then
  - Return ⊥
- else if $(N, C, \sigma) \in T$ then
  - Return ⊥

- $v \leftarrow \text{VRFY}(k_3, C || N, \sigma)$
- if $v = 0$ then
  - Return ⊥
- end if

- $IV \leftarrow F_{k_2}^*(N)$
- $M \leftarrow D_{k_1}^IV(C)$
- Return $M$

proc $\text{Enc}(N, M, l_F, l_E, l_T)$:

- if $(l_F, l_E, l_T) \notin \mathcal{L}_{\text{enc}}$ then
  - Return ⊥
- end if

- $IV \leftarrow F_{k_2}^*(N)$
- $C \leftarrow E_{k_1}^IV(M)$
- $\sigma \leftarrow \text{Tag}(k_3, C || N)$
- $\Lambda_F \leftarrow l_F(k_2, N, R_F)$
- $\Lambda_E \leftarrow l_E(k_1, M, IV, R_E)$
- $\Lambda_T \leftarrow l_T(k_3, C || N, R_T)$
- $T \leftarrow T \cup \{(N, C, \sigma)\}$
- Return $(C, \sigma, \Lambda_F, \Lambda_E, \Lambda_T)$

proc $\text{Dec}(N, C, \sigma, l_V, l_F, l_D)$:

- if $(l_V, l_F, l_D) \notin \mathcal{L}_{\text{dec}}$ then
  - Return ⊥
- end if

- if $(N, C, \sigma) \in \mathcal{U}$ then
  - Return ⊥
- else if $(N, C, \sigma) \in S$ then
  - Return ⊥
- else if $(N, C, \sigma) \in \mathcal{V}$ then
  - Return ⊥
- end if

- $v \leftarrow \text{VRFY}(k_3, C || N, \sigma)$
- $\Lambda_V \leftarrow l_V(k_3, C || N, \sigma)$
- $\mathcal{U} \leftarrow \mathcal{U} \cup \{(N, C, \sigma)\}$
- if $v = 0$ then
  - Return $\Lambda_V$
- end if

- $IV \leftarrow F_{k_2}^*(N)$
- $M \leftarrow D_{k_1}^IV(C)$
- $\Lambda_F \leftarrow l_F(k_2, N, R_F)$
- $\Lambda_D \leftarrow l_D(k_1, IV, C, R_D)$
- Return $(M, \Lambda_V, \Lambda_F, \Lambda_D)$

Figure 5.35.: The game $G_a$, used to prove nALE security of A5
experiment $\text{Exp}^{G_b}_{A}(A)$:

$S \leftarrow \{\}$
$T \leftarrow \{\}$
$U \leftarrow \{\}$
$V \leftarrow \{\}$
$k_1 \xleftarrow{\$} KG$
$k_2 \xleftarrow{\$} KG$
$k_3 \xleftarrow{\$} KG$
$b' \leftarrow A^{\text{RR}()}\text{, }\text{RP}()\text{, }\text{Enc}()\text{, }\text{Dec}()$

Return $b'$

proc $\text{RR}(N, M)$:
$IV \leftarrow F_{k_2}^*(N)$
$C \leftarrow \mathcal{E}_{k_1}^I(M)$
$\sigma \leftarrow \text{Tag}(k_3, C||N)$
$S \leftarrow S \cup \{(N, C, \sigma)\}$

Return $(C, \sigma)$

proc $\text{RP}(N, C, \sigma)$:
$V \leftarrow V \cup \{(N, C, \sigma)\}$

Return ⊥

proc $\text{Enc}(N, M, l_F, l_E, l_T)$:

if $(l_F, l_E, l_T) \notin \mathcal{L}_{\text{enc}}$ then

Return $\|$s

eend if

$\text{proc} \text{Dec}(N, C, \sigma, l_V, l_F, l_D)$:

if $(l_V, l_F, l_D) \notin \mathcal{L}_{\text{dec}}$ then

Return $\|$s

eend if

if $(N, C, \sigma) \in V$ then

Return $\|$s

e else if $(N, C, \sigma) \in S$ then

Return $\|$s

e end if

$v \xleftarrow{\$} \text{VRFY}(k_3, C||N, \sigma)$
$\Lambda_V \leftarrow l_V(k_3, C||N, \sigma, R_V)$
$U \leftarrow U \cup \{(N, C, \sigma)\}$

if $v = 0$ then

Return $\Lambda_V$

e end if

$IV \xleftarrow{\$} F_{k_2}^*(N)$
$M \xleftarrow{\$} D_{k_1}^E(C)$
$\Lambda_F \leftarrow l_F(k_2, N, R_F)$
$\Lambda_D \leftarrow l_D(k_1, IV, C, R_D)$

Return $(M, \Lambda_V, \Lambda_F, \Lambda_D)$

Figure 5.36.: The game $G_b$, used to prove nALE security of A5

184
experiment $\text{Exp}^{G_c}_{\mathcal{A}}(A)$:

$S \leftarrow \{\}$
$T \leftarrow \{\}$
$\mathcal{U} \leftarrow \{\}$
$\mathcal{V} \leftarrow \{\}$
$k_1 \leftarrow KG$
$k_2 \leftarrow KG$
$k_3 \leftarrow KG$
b' \leftarrow A^{\text{RR}(\cdot),\text{RP}(\cdot),\text{Enc}(\cdot),\text{Dec}(\cdot)}$

Return $b'$

proc $\text{RR}(N, M)$:

if $F[N] = \perp$ then
    $F[N] \leftarrow \{0, 1\}^k$
end if

$IV \leftarrow F[N]$
$C \leftarrow E^{IV}_{k_1}(M)$
$\sigma \leftarrow \text{Tag}(k_3, C || N)$

$S \leftarrow S \cup \{(N, C, \sigma)\}$

Return $(C, \sigma)$

proc $\text{RP}(N, C, \sigma)$:

$\mathcal{V} \leftarrow \mathcal{V} \cup \{(N, C, \sigma)\}$

Return $\perp$

proc $\text{Enc}(N, M, l^F, l^E, l^T)$:

if $(l^F, l^E, l^T) \notin \mathcal{L}_{\text{enc}}$ then
    Return $\perp$
end if

$IV \leftarrow F^*_{k_2}(N)$
$C \leftarrow E^{IV}_{k_1}(M)$
$\sigma \leftarrow \text{Tag}(k_3, C || N)$
$\Lambda_F \leftarrow l_F(k_2, N, R_F)$
$\Lambda_E \leftarrow l_E(k_3, M, IV, R_E)$
$\Lambda_T \leftarrow l_T(k_3, C || N, R_T)$
$\mathcal{T} \leftarrow \mathcal{T} \cup \{(N, C, \sigma)\}$

Return $(C, \sigma, \Lambda_F, \Lambda_E, \Lambda_T)$

proc $\text{Dec}(N, C, \sigma, l^V, l^F, l^D)$:

if $(l^V, l^F, l^D) \notin \mathcal{L}_{\text{dec}}$ then
    Return $\perp$
end if

if $(N, C, \sigma) \in \mathcal{V}$ then
    Return $\perp$
else if $(N, C, \sigma) \in S$ then
    Return $\perp$
end if

$v \leftarrow \text{VRFY}(k_3, C || N, \sigma)$
$\Lambda_V \leftarrow l_V(k_3, C || N, \sigma)$
$\mathcal{U} \leftarrow \mathcal{U} \cup \{(N, C, \sigma)\}$

if $v = 0$ then
    Return $\Lambda_V$
end if

$IV \leftarrow F^*_{k_2}(N)$
$M \leftarrow D^V_{k_1}(C)$
$\Lambda_F \leftarrow l_F(k_2, N, R_F)$
$\Lambda_D \leftarrow l_D(k_1, IV, C, R_D)$

Return $(M, \Lambda_V, \Lambda_F, \Lambda_D)$

Figure 5.37.: The game $G_c$, used to prove nALE security of A5
experiment $\text{Exp}_{\text{A}}^{\text{Gd}}(A)$:

$S \leftarrow \{\}$
$T \leftarrow \{\}$
$U \leftarrow \{\}$
$V \leftarrow \{\}$
$k_1 \overset{\$}{\leftarrow} KG$
$k_2 \overset{\$}{\leftarrow} KG$
$k_3 \overset{\$}{\leftarrow} KG$

$b' \leftarrow A^{\text{RR}(), \text{RP}(), \text{Enc}(), \text{Dec}()}$
Return $b'$

proc $\text{RR}(N, M)$:
if $F[N] = \perp$ then
$F[N] \leftarrow \{0, 1\}^*$
end if
$IV \leftarrow F[N]$
if $E[M, IV] = \perp$ then
$l \leftarrow |E^IV_1(M)|$
$E[M, IV] \leftarrow \{0, 1\}^l$
end if
$C \leftarrow E[M, IV]$
$\sigma \leftarrow \text{Tag}(k_3, C||N)$
$S \leftarrow S \cup \{(N, C, \sigma)\}$
Return $(C, \sigma)$

proc $\text{RP}(N, C, \sigma)$:
$V \leftarrow V \cup \{(N, C, \sigma)\}$
Return $\perp$

proc $\text{Enc}(N, M, l_F, l_E, l_T)$:
if $(l_F, l_E, l_T) \notin \mathcal{L}_{\text{enc}}$ then
Return $\perp$
end if
$IV \leftarrow F_{k_2}(N)$
$C \xrightarrow{\text{RE}} E^IV_{k_1}(M)$
$\sigma \leftarrow \text{Tag}(k_3, C||N)$
$A_F \leftarrow l_F(k_2, N, R_F)$
$\Lambda_E \leftarrow l_E(k_1, M, IV, R_E)$
$\Lambda_T \leftarrow l_T(k_3, C||N, R_T)$
$T \leftarrow T \cup \{(N, C, \sigma)\}$
Return $(C, \sigma, \Lambda_F, \Lambda_E, \Lambda_T)$

proc $\text{Dec}(N, C, \sigma, l_V, l_F, l_D)$:
if $(l_V, l_F, l_D) \notin \mathcal{L}_{\text{dec}}$ then
Return $\perp$
end if
if $(N, C, \sigma) \in V$ then
Return $\perp$
else if $(N, C, \sigma) \in S$ then
Return $\perp$
end if
$v \leftarrow \text{VRFY}(k_3, C||N, \sigma)$
$\Lambda_V \leftarrow l_V(k_3, C||N, \sigma, R_V)$
$U \leftarrow U \cup \{(N, C, \sigma)\}$
if $v = 0$ then
Return $\Lambda_V$
end if
$IV \leftarrow F_{k_2}(N)$
$M \leftarrow D^IV_{k_1}(C)$
$A_F \leftarrow l_F(k_2, N, R_F)$
$\Lambda_D \leftarrow l_D(k_1, IV, C, R_D)$
Return $(M, \Lambda_V, \Lambda_F, \Lambda_D)$

Figure 5.38.: The game $G_d$, used to prove nALE security of A5

186
experiment $\text{Exp}_{A}^{G}(A)$:
$S \leftarrow \{\}$
$T \leftarrow \{\}$
$U \leftarrow \{\}$
$V \leftarrow \{\}$
$k_1 \overset{\$}{\leftarrow} KG$
$k_2 \overset{\$}{\leftarrow} KG$
$k_3 \overset{\$}{\leftarrow} KG$
$b' \leftarrow A_{\text{RR}(\cdot), \text{RP}(\cdot), \text{Enc}(\cdot), \text{Dec}(\cdot)}$
Return $b'$

proc $\text{RR}(N, M)$:
if $F[N] = \bot$
then
$F[N] \overset{\$}{\leftarrow} \{0,1\}^\kappa$
def
$IV \leftarrow F[N]$
if $E[M, IV] = \bot$
then
$l \leftarrow |E_{k_1}^IV(M)|$
$E[M, IV] \overset{\$}{\leftarrow} \{0,1\}^l$
def
$C \leftarrow E[M, IV]$
if $T[C||N] = \bot$
then
$T[C||N] \overset{\$}{\leftarrow} \{0,1\}^\kappa$
def
$\sigma \leftarrow T[C||N]$
$S \leftarrow S \cup \{(N, C, \sigma)\}$
Return $(C, \sigma)$

proc $\text{RP}(N, C, \sigma)$:
$V \leftarrow V \cup \{(N, C, \sigma)\}$
Return $\bot$

proc $\text{Enc}(N, M, l_F, l_E, l_T)$:
if $(l_F, l_E, l_T) \notin L_{\text{enc}}$
then
Return $\bot$
def
$IV \overset{R_V}{\leftarrow} F_{k_2}^* (N)$
$C \overset{R_E}{\leftarrow} C_{k_1}^IV (M)$
$\sigma \overset{R_T}{\leftarrow} \text{Tag}(k_3, C||N)$
$\Lambda_F \leftarrow l_E(k_2, N, R_F)$
$\Lambda_E \leftarrow l_E(k_1, M, IV, R_E)$
$\Lambda_T \leftarrow l_T(k_3, C||N, R_T)$
$T \leftarrow T \cup \{(N, C, \sigma)\}$
Return $(C, \sigma, \Lambda_F, \Lambda_E, \Lambda_T)$

proc $\text{Dec}(N, C, \sigma, l_V, l_F, l_D)$:
if $(l_V, l_F, l_D) \notin L_{\text{dec}}$
then
Return $\bot$
def
if $(N, C, \sigma) \in V$
then
Return $\bot$
else if $(N, C, \sigma) \in S$
then
Return $\bot$
def
$v \overset{R_V}{\leftarrow} \text{VRFY}(k_3, C||N, \sigma)$
$\Lambda_V \leftarrow l_V(k_3, C||N, \sigma, R_V)$
$U \leftarrow U \cup \{(N, C, \sigma)\}$
if $v = 0$
then
Return $\Lambda_V$
def
$IV \overset{R_F}{\leftarrow} F_{k_2}^* (N)$
$M \overset{R_D}{\leftarrow} D_{k_1}^IV (C)$
$\Lambda_F \leftarrow l_F(k_2, N, R_F)$
$\Lambda_D \leftarrow l_D(k_1, IV, C, R_D)$
Return $(M, \Lambda_V, \Lambda_F, \Lambda_D)$

Figure 5.39.: The game $G_e$, used to prove nALE security of A5

187
**Considering Weaker Security Notions**  In this work, we have been considering the nALE security definition. It remains an open question if generic composition is possible, if the leakage is weakened to either L1 or L0. Since the attack on the 6 other generic composition schemes relied on the power of the L2 leakage, it is possible that these schemes are secure under L1 or L0 leakage. The L2 leakage has been shown to be extremely powerful for breaking schemes such as the PRLF in two shares, where the adversary can get leakage after receiving a challenge. The weaker notions correspond to the case where an adversary has physical access to a device to measure leakage but when the challenge is seen they no longer have access to the device. This model may enable more efficient schemes to be constructed.

### 5.6. Practical Considerations

In this section we focus on some practical considerations: how efficient are our schemes in comparison to other leakage resilient constructions in the literature, and what would a practical implementation need to guarantee it meets our leakage bound/assumptions?

Before giving a comparison we need to make some choices for parameters of the various schemes. In the case of schemes which have an underlying primitive of a PRF, we chose to instantiate this PRF with AES-128. AES-128 is chosen as it represents the most commonly deployed primitive. For our own scheme, and a somewhat comparable signature scheme, to instantiate the bilinear map, we use a pairing defined over a suitable pairing friendly elliptic curve. In this case we choose as group size parameter $2^{160}$, again with the motivation to reflect a state-of-the art security bound (i.e. $2^{80}$ is regarded as minimum for bound which is implied by a group of double this size).

#### 5.6.1. Security Considerations

Plugging in the selected group size of $2^{160}$ into the bounds shown within the theorems given above, shows that our schemes can tolerate approximately 35-50 bits of leakage per leakage function. This means 70-100 bits of leakage per function call for those schemes shared in two (which we will focus on for the remainder of this section) while three shares allows 105-150
bits of leakage per function call. The majority of our schemes can leak 35 bits, since they can approximately leak the logarithm of the group size over four. While the second MAC can leak approximately 50 bits, due to being able to leak a third of the logarithm of the group size. For a practical implementation it is important to note, the initial sharing of the secret key cannot leak any information. Consequently, it has to be assumed that this can be done in a secure environment by a suitably trusted party.

When considering what ‘35 bits of leakage’ means for our constructions from a practical perspective, it helps to think about the concrete instantiation of the scheme. As mentioned above, this means that the scheme will be using a pairing $e$ over a pairing friendly elliptic curve. Consequently $S_i^\bullet, S_i^\circ, R_i^\circ$ are elliptic curve points in $G_1$. The updating of the key corresponds to elliptic curve addition in $G_1$ while the recombination, after the pairing has been performed, is a multiplication in $G_3$. Given this concrete instantiation of our scheme, it is now easier to argue about which implementation decisions will provide the best protection. Since each function call can only leak a bounded number of bits, SPA resistance must be considered to stop the secrets leaking out within a single operation (and to help the device meet the theoretical leakage bound). This will probably take the form of blinding all elliptic curve points which correspond to the secret information (primarily the shares of the key and any randomness used). Blömer et al. [29] give an overview on various side channel attacks against pairings. From this it can be concluded that the pairing itself should also be implemented on blinded elliptic curve points.

If SPA protection still needs to be applied to the scheme, the question arises; what has been gained by constructing a provably secure scheme? Firstly, the scheme will be inherently protected from DPA attacks due to the constant updating of the key’s representation. Secondly, for the non-leakage resilient version, any leakage on the secret values should allow lattice based attacks, such as those by Nguyen and Shparlinski [124]. The leakage resilient version guarantees that no such attacks can succeed.

5.6.2. Comparison with Other Leakage Resilient Schemes

For comparisons we will mainly focus on authentication schemes but will also briefly touch on a few encryption schemes. Our metric for comparison will be the number of elliptic curve or
AES operations required by the functions. We will also report on the tolerated leakage, the key size and the output size. Table 5.1 gives an overview of this information for the authentication schemes, while Table 5.2 gives the information for encryption schemes. Below we explain the information given in the tables in more detail. As can be seen from these results, both our MAC and encryption schemes are highly competitive. We do not discuss our PRLF here, as due to its similarity to the MAC, similar results apply.

**Authentication Comparison**

Pereira *et al.* [131] propose a leakage resilient MAC in the simulatable leakage model. However, it will not be discussed in further detail here due to the work by Galea *et al.* [108] showing that no efficient simulator currently exists and the fact that it requires access to a leak free block cipher. It is not clear how such a block cipher could be implemented such that it does not leak. If it is possible to implement a block cipher in a leakage free manner it is not clear how many orders of magnitude slower than an unprotected block cipher it would be. This makes it hard to compare to other schemes (assuming it is possible to construct) and thus will not be included in the comparisons.

Hazay *et al.* [89] produce a leakage resilient MAC from extremely minimal assumptions (the existence of a one-way function). The disadvantage of this scheme is that it is in the bounded leakage model. Therefore, it may only leak a bounded total number of bits. From a practical perspective this implies that regardless of the number of times the device is used, it leaks the same amount. This of course does not relate to how a physical instantiation behaves, where a device leaks more, the more it is used. Instantiating the PRF with AES-128 and using the equations given in Theorem 5.6 (from [89]), AES will be called 512 times per tag and verify query. While this makes the scheme computationally expensive, a greater issue is the key size. The key must be of size approximately $2^{18}$ bits (32 KB). This will make the scheme impractical for many applications involving lightweight devices, such as smartcards. Under these parameters, the MAC is allowed to leak only a total of 512 bits (not per tag or verify call) which is fairly little compared to the size of the secret key.

Schipper provides a leakage resilient MAC in his Master’s thesis [145]. However, it is built
upon a EUF-CMA MAC and a PRLF. Hence the timing will be similar to those of the PRLFs which are investigated below. Therefore, explicit results for Schipper’s construction are not included here.

Due to the small number of leakage resilient MACs in the literature we will also provide comparisons against PRLFs and leakage resilient signature schemes. Leakage resilient signature schemes can be used as leakage resilient MACs by keeping the public key secret. Tag will then have the same leakage properties as sign, while verify will be able to leak arbitrarily, because even if the signature scheme’s public key leaks the scheme will be secure. Since in the proof of security for the signature scheme the public key is known by the adversary. While it is not immediately apparent how to construct a leakage resilient MAC from a PRLF, it seems feasible and thus the comparison is worth making.

The PRLF by Dodis and Pietrzak [56] allows for a continual amount of leakage (bounded per call) but the leakage functions must be fixed prior to the start of the security game and cannot be changed adaptively. They define the PRLF $\Gamma^F : \Sigma^{3\cdot \kappa+n} \times \Sigma^n \rightarrow \Sigma^{4\cdot \kappa+2\cdot n}$ created from a wPRF $F : \Sigma^{\kappa} \times \Sigma^n \rightarrow \Sigma^{4\cdot \kappa+2\cdot n}$. If $F$ is instantiated with AES-128 as follows; $F_k(x) = AES_k(x||000)||AES_k(x||001)||AES_k(x||010)||AES_k(x||011)||AES_k(x||100)||AES_k(x||101)$

and then take the desired number of output bits, this results in $\kappa = m = 128$, $n = 125$ and $\Gamma^F : \Sigma^{509} \times \Sigma^{128} \rightarrow \Sigma^{768}$. From this, each time the PRLF is called, $F$ is called $m + 1 = 129$ times and thus AES will be called 774 times per PRLF call. Even if $\Gamma^F$ can be converted into a MAC ‘for free’ it will still have a high overhead cost. This schemes leakage is directly related to the security of the wPRF. Assuming the wPRF provides 128 bits of security, the scheme can leak approximately 22 bits per call (as the scheme is allowed to leak a sixth of the bits of security provided by the wPRF).

Faust et al. [67] construct a more efficient PRLF than that of Dodis and Pietrzak. They construct a scheme $\Gamma^{F,m} : \{0, 1\}^{\kappa+(m+1)\cdot l} \times \{0, 1\}^m \rightarrow \{0, 1\}^n$ which uses the wPRF $F : \{0, 1\}^{\kappa} \times \{0, 1\}^l \rightarrow \{0, 1\}^{2\cdot \kappa}$ and $m + 1$ public random values of length $l$. If the wPRF is instantiated with AES-128 to get $F_k(x) = AES_k(x||0)||AES_k(x||1)$, meaning $m = \kappa = 128$, $l = 127$, $n = 256$ and $\Gamma^{F,128} : \{0, 1\}^{16511} \times \{0, 1\}^{128} \rightarrow \{0, 1\}^{256}$. Per invocation of the PRLF, AES is called 258 times. While this is an improvement, it is still a larger overhead.
However, more restricting is that, while the leakage is still continuous, both the inputs and leakage functions for the PRLF must be chosen non adaptively before the security game begins.

The signature scheme by Faust et al. [66] uses \(2 \cdot l\) exponentiations, \(4 \cdot (l - 1)\) multiplications, \(l - 1\) additions and 2 hash function calls in the signing algorithm and \(t \cdot l\) exponentiations, \(t \cdot l\) multiplications and \(t\) hash function calls in the verification algorithm. Where \(l\) is related to the underlying \(l\)-representation problem [35] (assumed to be hard) and \(t\) is the depth of the signature chain. The length of the chain is related to how many messages the construction can sign (and to a lesser extent, how many messages have been signed before the current message).

The downside of this scheme is that even if \(l = 2\), while signing is efficient, verification takes longer depending how deep the signature chain is. This could mean that verification quickly becomes too expensive for an embedded device to perform.

Galindo and Vivek [75] produce a signature scheme in the continuous leakage model with comparable results to our MAC. This is to be expected since they are both built upon the same key update mechanism. For signing, the algorithm uses 2 elliptic curve scalar multiplications, 5 elliptic curve additions and also generates a random curve point (and its inverse), while verification uses 2 elliptic curve point additions, 1 elliptic curve scalar multiplication and 2 pairings. Since a pairing is currently only slightly more expensive than an exponentiation, our tagging algorithm will be almost equivalent in timing to their signing algorithm. While their verification algorithm is faster than ours, their keys are larger and, therefore, the slower verification may be an acceptable trade off for the smaller key size. They assume the message comes from \(\mathbb{Z}_p\) (so that it can be used as an exponent), while in reality the message would be arbitrary bit string which would be hashed down to \(\mathbb{Z}_p\) adding the cost of a hash function to the timing cost. Their theorem states that they are allowed less than 80 bits of leakage, although the proof hides a lot of constants so this could in fact be considerably lower.

There are three other leakage resilient signatures in the literature [74, 159] based on the Schnorr signature scheme [147], Boneh-Lynn-Shacham (BLS) short signature scheme [34] and Waters signature scheme [166] respectively (the second paper contains two signature schemes). Due to the similarity of techniques deployed as the Galindo and Vivek signature scheme above (they both use the same key update mechanism), we will not discuss the schemes in detail here.
However, they are included in the table for comparison.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Leakage</th>
<th>Key Size</th>
<th>Tag Size</th>
<th>Tag Time</th>
<th>Verification Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our scheme</td>
<td>( O(\log \epsilon - 1 - n/4) ) for ( \epsilon = \text{Adv}_{wprf}(A) )</td>
<td>2 EC points (approx 320 bits)</td>
<td>1 EC point (approx 160 bits)</td>
<td>1 hash</td>
<td>512 AES calls</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HLWW: [89]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dp: [56]</td>
<td>( O(\log \epsilon - 1 - n/4) ) for ( \epsilon = \text{Adv}_{wprf}(A) )</td>
<td>509 bits</td>
<td>762 bits</td>
<td>1 hash</td>
<td>774 AES calls</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FP: [67]</td>
<td>( O(\log \epsilon - 1 - n/4) ) for ( \epsilon = \text{Adv}_{wprf}(A) )</td>
<td>128 bits</td>
<td>16,383 public bits</td>
<td>125 bits</td>
<td>258 AES calls</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>FKPR: [66]</td>
<td>( O(\log \epsilon - 1 - n/4) ) (approx 40 bits)</td>
<td>Varies</td>
<td>Varies</td>
<td>2 hashes</td>
<td>Varies</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GV: [74]</td>
<td>(&lt; &lt; \log p &gt; p ) is group size (approx 60 bits)</td>
<td>sk: 2 EC points, pk: 3 EC points (approx 800 bits)</td>
<td>2 EC points (approx 320 bits)</td>
<td>1 hash</td>
<td>1 hash</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>GV: [75]</td>
<td>(&lt; &lt; \log p &gt; p ) is group size (approx 60 bits)</td>
<td>sk: 2 EC points, pk: 1 EC point (approx 480 bits)</td>
<td>1 EC point, 1 element of ( \mathbb{Z}_p ) (approx 320 bits)</td>
<td>1 hash</td>
<td>1 hash</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLNL-1: [159]</td>
<td>(&lt; &lt; \log p &gt; p ) is group size (approx 60 bits)</td>
<td>sk: 2 EC points, pk: 1 EC point (approx 480 bits)</td>
<td>2 EC points (approx 320 bits)</td>
<td>1 hash</td>
<td>1 hash</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TLNL-2: [159]</td>
<td>(&lt; &lt; \log p &gt; p ) is group size (approx 60 bits)</td>
<td>sk: 2 EC points, pk: ( n + 2 ) EC points, ( n ) length of messages (approx 160*(( n + 4 ) bits)</td>
<td>2 EC points (approx 320 bits)</td>
<td>2 scalar multiplications ( n + 4 ) EC additions ( n ) random integer generated</td>
<td>2 pairings</td>
</tr>
</tbody>
</table>

Table 5.1.: A comparison of possible authentication schemes

**Implementation of the MAC** In the published version of this work [115] we implement our MAC so we can further investigate its performance. The (algorithmic) implementation decisions will be discussed below. By implementing the MAC in software and running it on a small device we were able to obtain concrete timing results for both the leakage and non-leakage version, showing how much the leakage resilience slowed down the scheme. Since we had a physical instantiation of the scheme we were then able to try and attack it using standard side channel attacks. For more details see the published version [115].
Encryption Comparison

Pereira et al. [131] also propose a leakage resilient encryption scheme in the simulatable leakage model built from the leakage resilient PRG in the same model [158]. Due to the same reasons above (the use of a leak free block cipher and no efficient simulator known to exist) we do not discuss this encryption scheme further.

As previously discussed, due to the non-adaptiveness of the leakage functions (and input) to the PRLFs [56, 67], it is not immediately apparent how the PRLFs can be used within a mode of operation, to construct an encryption scheme. However, since there may be a way to achieve this for a certain class of leakage (it would have to be non adaptive and from a restrictive set), it still makes an interesting comparison. The amount of work and leakage available would be the same as above but for each block of the message. Therefore, the PRLFs discussed in the previous section are included in the table for comparison.

Dziembowski and Pietrzak [65] construct a leakage resilient PRG, which it may be possible to convert into a stream cipher. The current issue is being able to perform this conversion in the adaptive leakage setting. The non-leakage process to use a PRG as a stream cipher is to run the PRG with the key on a random input and use this as a keystream to exclusive-or to the message. However, by repeatedly encrypting messages, the initial key would be able to be repeatedly leaked upon until it is known to the adversary. Hence an alternative method would have to be used to convert the leakage resilient PRG into a leakage resilient encryption scheme. This scheme uses an alternating structure with two keys such that each key is used (and updated) independently. Each iteration uses a PRG and an extractor to calculate the next output block. The downside of this construction is that extractors are reasonably inefficient and have been shown to leak a fair amount, especially in the profiled case [118].

Pietrzak [133] improves upon the previous PRG by replacing the extractor and PRG with a wPRF. While this scheme has the same issues converting it into an encryption scheme as the previous PRG, it will be more efficient. The adversarial advantage against the PRG is approximately the twelfth root of the adversarial advantage against the wPRF. Therefore, for the scheme to provide 80 bits of security, an approximately 960 bit wPRF is required which would invoke AES approximately 8 times. While this scheme is competitive, it is not clear how to secure it
against repeat use without a leak free component. Figure 6.15 in Sect. 6.6 gives a diagram of the construction, where the scheme is discussed in more detail.

There have been many leakage resilient encryption schemes in the bounded leakage model [89, 104, 137]. However, due to the weaker leakage model, the efficiency comparison between these schemes and the one presented above are not valid. Thus we will not discuss these schemes in detail.

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Leakage</th>
<th>Key Size</th>
<th>Ciphertext Size</th>
<th>Encryption Time</th>
<th>Decryption Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our scheme</td>
<td>$\mathcal{O}(\log(\frac{1}{\epsilon} - 1))$ for $\epsilon = \text{Adv}_{\text{cwecma}}(A)$</td>
<td>2 EC points (approx 320 bits)</td>
<td>$m + 1$ EC points (approx 160$(m + 1)$ bits)</td>
<td>$m$ hashes</td>
<td>$m$ hashes</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$m$ block messages</td>
<td>$m$ messages</td>
<td>$m$ messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$2 \cdot m$ pairings</td>
<td>$2 \cdot m$ pairings</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$3 \cdot m$ EC additions</td>
<td>$3 \cdot m$ EC additions</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$m$ random points (and inverses) generated</td>
<td>$m$ random points (and inverses) generated</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$m$ block messages</td>
<td>$m$ block messages</td>
</tr>
<tr>
<td>DP [56]</td>
<td>$\mathcal{O}(\log(\frac{1}{\epsilon} - 1))$ for $\epsilon = \text{Adv}_{\text{cwecma}}(A)$</td>
<td>509 bits</td>
<td>762 - $m$ bits</td>
<td>$774 \cdot m$ AES calls</td>
<td>$774 \cdot m$ AES calls</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$m$ block messages</td>
<td>$m$ block messages</td>
<td>$m$ block messages</td>
</tr>
<tr>
<td>FPS [67]</td>
<td>$\mathcal{O}(\log(\frac{1}{\epsilon} - 1))$ for $\epsilon = \text{Adv}_{\text{cwecma}}(A)$</td>
<td>128 bits</td>
<td>16,383 public bits</td>
<td>$256 \cdot m$ AES calls</td>
<td>$256 \cdot m$ AES calls</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$m$ block messages</td>
<td>$m$ block messages</td>
<td>$m$ block messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$m$ PRG calls</td>
<td>$m$ PRG calls</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$m$ extractor calls</td>
<td>$m$ extractor calls</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$m$ block messages</td>
<td>$m$ block messages</td>
</tr>
<tr>
<td>DP [65]</td>
<td>$\mathcal{O}(\log(\frac{1}{\epsilon} - 1))$ for $\epsilon = \text{Adv}_{\text{cwecma}}(A)$</td>
<td>256 secret bits</td>
<td>512 public bits</td>
<td>Same size as message</td>
<td>$m$ PRG calls</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$m$ block messages</td>
<td>$m$ block messages</td>
<td>$m$ block messages</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$m$ PRG calls</td>
<td>$m$ PRG calls</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$m$ extractor calls</td>
<td>$m$ extractor calls</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$m$ block messages</td>
<td>$m$ block messages</td>
</tr>
<tr>
<td>P [133]</td>
<td>$\mathcal{O}(\log(\frac{1}{\epsilon} - 1))$ for $\epsilon = \text{Adv}_{\text{cwecma}}(A)$</td>
<td>1920 bits</td>
<td>960 public bits</td>
<td>$960 \cdot m$ AES calls</td>
<td>$960 \cdot m$ AES calls</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>$m$ message blocks</td>
<td>$m$ message blocks</td>
<td>$m$ message blocks</td>
</tr>
</tbody>
</table>

Table 5.2.: A comparison of possible leakage resilient encryption schemes

### 5.6.3. Random Point Generation

Since all of the schemes, based on the KP Key Update Mechanism, require a lot of randomness to update the key, the question arises as to what is the best way to generate the randomness. This is a particularly important question since all schemes have a leakage bound which they must adhere to. To provide security it is important that the random point generation is within these bounds and leaves enough bits to allow the other parts of the algorithm to leak as well.

We investigate four common methods to generate random points on the elliptic curve (given in Fig. 5.3) and discuss how they would leak in practice.
Random Scalar:
Let $P$ be a generator of $E(\mathbb{F}_p)$
$$r \overset{\$}{\leftarrow} \mathbb{F}_p^*$$
return $R = [r]P$

Try_and_Increment:
$$x \overset{\$}{\leftarrow} \mathbb{F}_p^*$$
loop
if $x^2 + b$ is square then
$$i \overset{\$}{\leftarrow} \{0, 1\}$$
y $\leftarrow -1^i \cdot \sqrt{x^2 + b}$
Return $(x, y)$
else
$$x \leftarrow x + 1$$
end if
end loop

Random Sampler:
loop
$$x \overset{\$}{\leftarrow} \mathbb{F}_p^*$$
if $x^2 + b$ is square then
$$i \overset{\$}{\leftarrow} \{0, 1\}$$
y $\leftarrow -1^i \cdot \sqrt{x^2 + b}$
Return $(x, y)$
end if
end loop

SWEncoding:
$$t \overset{\$}{\leftarrow} \mathbb{F}_p^*$$
w $\leftarrow \sqrt{-3} \cdot t/(1 + b + t^2)$
$$x_1 \leftarrow -1 + \sqrt{-3/2 - tw}$$
$$x_2 \leftarrow -1 + x_1$$
$$x_3 \leftarrow 1 + 1/w^2$$
r_1, r_2, r_3 \overset{\$}{\leftarrow} \mathbb{F}_p^*$$
$$\alpha \leftarrow \chi_p(r_1^2 \cdot (x_1^3 + b))$$
$$\beta \leftarrow \chi_p(r_2^2 \cdot (x_2^3 + b))$$
i $\leftarrow [(\alpha - 1) \cdot \beta \mod 3] + 1$
$$R \leftarrow (x_i, \chi_p(r_3^3 \cdot t) \cdot \sqrt{x_i^3 + b})$$
Return $R$

Table 5.3.: The algorithms for the four random point generation algorithms considered

Random Scalar

The random scalar method is one of the most common methods used. It is how some of the leakage resilient papers explicitly state how to generate a random point [74, 75, 101, 159]. This method generates a random exponent and raises a public generator to the given exponent. However, it is well known that a double and an add look different within a power (or EM) trace, because they are calculated differently, and thus an adversary can completely recover the secret exponent and thus the random point [119]. Since the leakage bound is always less than a single elliptic curve point (otherwise the secret key could be leaked), this is too much leakage for an adversary to learn. It is clear that if the adversary is allowed to learn all randomness within a scheme plus a single bit of leakage on each share then they can completely recover the secret...
Figure 5.40.: EM traces for an RSA decryption on a Beagle Bone Black.

key. Hence, even though this is the most popular way of generating a random elliptic curve point, it should not be used when implementing a provably secure leakage scheme as it cannot support the bounds.

Figure 5.40(a) shows the result of performing an EM SPA attack against the right to left binary exponentiation of an RSA decryption. In this diagram, square and multiply look different and therefore the key can be extracted from the trace. Figure 5.40(b) shows the same trace annotated with the secret key. This exact process can also be performed on the scaler multiplication used on an elliptic curve, since addition and doubling look different.

**Incremental Rejection Sampling**

The incremental rejection sampling method chooses a random \( x \) and checks to see if it sits on the elliptic curve using the Jacobi symbol (discussed in more detail below). If the point sits on the curve it is used as the random point, otherwise it is repeatedly incremented until a value is found which does sit on the curve, which is then used as the given random point. Since several of the random point algorithms use the Jacobi symbol, the leakage will be discussed in more detail below. In this algorithm the increment and test step will be called a variable number of times and the number of iterations will depend on the initial point selected. From the leakage the adversary can determine how many times the point is incremented and tested, thus giving the adversary information about the selected point. The amount of leakage will depend on the
selected curve and the random point initially chosen but it is worth noting that this method does provide the adversary with additional information about the random point, even if just a few bits.

**Random Rejection Sampling**

This method is similar to the previous one where a random point is chosen and checked if it is on the curve. However, how the algorithm behaves when the value is not on the curves is different. If the point is not on the curve, instead of incrementing, it selects a new value independently at random. This method will again take a variable amount of time depending on the randomness selected. However, since each new piece of randomness is independent of the all previous randomness, the amount of time the algorithm takes to execute will not reveal any information about the final random point selected. The disadvantage of this approach is that it requires a lot of new randomness to operate but in return provides less leakage than the previous method.

**Shallue-Van de Woestijne Encoding [70]**

The Shallue-van de Woestijne encoding [152] has been used for cryptography by Fouque and Tibouchi [70] to hash to elliptic curves. It has been discussed in context of leakage resilient schemes by Galindo et al. [73]. The high level overview of this method is taking a constant time method of hashing an arbitrary string to an elliptic curve point. Thus, to generate an elliptic curve point, a random string is chosen and hashed to the curve. However, it has been shown that this does not generate a point uniformly on the curve. To resolve this it is suggested that the hash function be run twice, on two different strings, and that the combination of the two points provides the random point which is suitably close to uniform. Galindo et al. discuss traditional side channel countermeasures which may need to be applied to make this leak less. Firstly, a point inversion may leak a lot of information since the Extended Euclidean Algorithm (EEA) is fairly irregular [2,4,11]. To counter this, masking can be applied before the EEA and removed afterwards. If \( x \) is the element to be inverted, masking it with random \( r \) and running EEA gives \( (x \cdot r)^{-1} \) and then the mask can be removed by multiplying through by \( r \). Leakage on the Jacobi symbol will be discussed more below.
Choice of Random Point Sampler

From the four methods detailed above the random rejection sampling and Shallue-van de Woestijne encoding are the most desirable as they leak the least amount of information per random point generated. The only part of the computations that may leak any useful information is the Jacobi symbol calculation which will be discussed in more detail below. Out of these two most desirable options, the random rejection sampling is the most efficient and thus the one we would suggest using for sampling random points on an elliptic curve.

5.6.4. The Jacobi Symbol

The Jacobi symbol is used in three of the four randomness generation algorithms to determine if a value is square and thus sits on the curve. This determines if a valid point was chosen. For any integer \(a\) and positive odd integer \(n\), the Jacobi symbol is formally defined as:

\[
\left( \frac{a}{n} \right) = \left( \frac{a}{p_1} \right)^{\alpha_1} \cdots \left( \frac{a}{p_k} \right)^{\alpha_k}
\]

where \(n = p_1^{\alpha_1} \cdots p_k^{\alpha_k}\) for primes \(p_1, \ldots, p_k\) and \(\left( \frac{a}{p_i} \right)\) is the Legendre symbols, which given an integer \(a\) and odd prime \(p\) is defined as:

\[
\left( \frac{a}{p} \right) = \begin{cases} 
0 & \text{if } a = 0 \mod p \\
1 & \text{if } a \neq 0 \mod p \text{ and for some } x \in \mathbb{Z}, x^2 = a \mod p \\
-1 & \text{otherwise}
\end{cases}
\]

The Legendre symbol is calculated in the following manner:

\[
\left( \frac{a}{p} \right) = a^{\frac{p-1}{2}} \mod p
\]

Thus, if the prime factorisation of an integer \(n\) is known, the Jacobi symbol can be calculated by computing the Legendre symbols and taking the corresponding product. However, if the prime
factorisation is not known (it can be hard to calculate in cryptography due to the size of the
numbers - in fact sometimes security is based on this fact) then another method is required to
calculate the Jacobi symbol. Before the algorithm to compute the Jacobi symbol can be given,
some rules about the Jacobi symbol are required.

1. If \( n \) is prime then the Jacobi symbol \( \left( \frac{a}{n} \right) \) is simply the Legendre symbol and can be
evaluated as such

2. If \( a = b \mod n \) then \( \left( \frac{a}{n} \right) = \left( \frac{b}{n} \right) \)

3. \( \left( \frac{ab}{n} \right) = \left( \frac{a}{n} \right) \cdot \left( \frac{b}{n} \right) \)

4. \( \left( \frac{a}{n} \right) \cdot \left( \frac{n}{a} \right) = (-1)^{(a-1)(n-1)/4} \)

5. \( \left( \frac{2}{n} \right) = (-1)^{n^2-1} \)

The algorithm is then as follows:

1. Reduce the numerator modulo the denominator
2. Extract (and evaluate) any factors of two from the numerator
3. If possible, evaluate (e.g. prime denominator, non-coprime values)
4. Switch the numerator and denominator and repeat from step one

**Leakage on the Jacobi Symbol**

Now that the calculation of the Jacobi symbol has been described it is possible to discuss how it
leaks, what can be learnt from the leakage and how the leakage can be prevented.

From the description of the algorithm it can be seen that the algorithm does not run a fixed
number of times but runs until certain exit conditions are met. Since in the calculation of \( \left( \frac{a}{n} \right), n \)
is a public fixed parameter while \( a \) is secret, the run time will completely depend on \( a \).

To discover how much information can be extracted from a power trace for the Jacobi symbol
we ran the Jacobi symbol on a variety of different inputs on an embedded platform. Figure 5.41
Figure 5.41.: Power traces for the Jacobi symbol [115]

shows the leakage for two different inputs which have 99 and 96 conditional switches respectively. We were not able to exploit the differences within these traces. Known inputs which perform an unusually high (or low) number of conditional switches may be able to be distinguished from other inputs. We leave it as further work to try and discover and exploit such information.

One approach to prevent leakage is using blinding. To test if the value $x$ is square, choose a random value $r$ and test whether $x \cdot r^2$ is square. Since $x$ is being blinded by a square value, $x \cdot r^2$ will be square if, and only if, $x$ is square. However, since $x$ is being blinded by a randomly selected square, the leakage will now correspond to this point, instead of $x$, making it considerably harder to extract any information about $x$.
5.6.5. Hash Functions

A similar issue that occurred for random point generation also occurs when a value must be hashed to the curve. With slight tweaks some of the algorithms above can be used. The scalar multiplication algorithm can be used but instead of choosing a random scalar, a hash function $H : \{0, 1\}^* \rightarrow \mathbb{Z}_p$ is used to generate the scalar. This makes the resulting elliptic curve point $[H(M)]P$ for message $M$. This method is susceptible to the same issues as the scalar multiplication for random point. The incremental rejection sampling can be used to hash a message to the curve. This works by hashing the message to a value in $\mathbb{Z}_p$ and then using this as the initial $x$ in the increment step. This is deterministic since, given a deterministic hash, it will always take the same number of steps to find a valid point on the curve. Again, the number of increments to the value will leak information about the hash of the message. Both of the above require the hash $H$ to construct the hash to the curve and thus it must also be taken into consideration how $H$ leaks. The rejection sampler cannot be converted into a hashing method. Shallue-van de Woestijne encoding was designed for hashing a message to the curve and not for randomness generation. Thus this can be used by passing the message in, instead of a random value. For hashing we would recommend using Shallue-van de Woestijne encoding, as it is the one, out of the three options, which leaks less.

If instead an elliptic curve point needs to be hashed to a string, this is more straightforward and standard techniques can be used. A suitable string representation of the elliptic curve point (i.e. the binary representation of the $x$ coordinate and a bit to select the $y$ coordinate) can be used and a standard (side channel protected), string to string, hash function can be applied.

5.7. Conclusion

In this chapter we have added leakage resilient MACs, a PRF and varying level security encryption schemes to the known literature, all of which are based on the same underlying key update mechanism. We then concluded the chapter by analysing the efficiency of our schemes against other schemes from the literature and discussing what practical considerations need to be made such that the instantiation of the scheme meets all of the requirements as defined by the
Theoretical analysis.

The literature already contains a leakage resilient KEM and signature schemes, and combined with the work given here there would be enough for a leakage resilient cryptography suite which could be used within Secure Sockets Layer (SSL), or similar, when the extra security is required.

The advantage of building all the schemes from the same key update function is simplicity. This reduces the number of components that need to be implemented and, more importantly, reduces the number of components which require engineering countermeasures (such as masking) applied to them. If at any stage the key update mechanism can be implemented in a new manner, such that it leaks less information, then all schemes built upon it will inherently leak less information.

Future work in this area includes; analysing the key update mechanism in more detail from a practical perspective to see how it can be implemented as to leak as little information as possible. Taking the suite of leakage resilient schemes and implementing them into something similar to SSL to see how they behave when used together, instead of independently, would also be a worthwhile exercise. Finally can any theoretical statements be made about using the schemes as a suite to back up the practical findings.
6. Practically Estimating $\lambda$ Bits of Leakage

The key rank and key enumeration work, presented in this chapter, was joint with Mr Jonathan O’Connell, Professor Elisabeth Oswald and Dr Martijn Stam [113]. I worked on the algorithm design (both for key rank and enumeration) and, where necessary, helped with the implementation of the algorithm, including providing initial working code of the algorithms. Due to this, some of the graphs given within this chapter are from the paper [113]. The graphs relating to the running of the algorithm, given in the implementation section, were produced by Jonathan. Any figures explaining the algorithm were produced by myself. The work at the end of this chapter, using key rank to measure the leakage of a device, was performed solely by me.

The continuous leakage model has several advantages, including the guarantee that if the leakage per call to the device is bounded, then the device will be secure regardless of how many times it is called. One of the biggest issues with the continuous leakage model (and several of the other leakage models) is the concrete bound $\lambda$ in the number of bits that can be leaked per leakage function. The theoretical community has not created a metric, to complement the model, to measure how many bits a device actually leaks. There has been work [73, 115] which discusses the best implementation decisions to make when implementing given schemes. While the decisions made in these works clearly make the implementation harder to attack, there has still been no way given to measure the amount of bits leaked in these particular implementations.

It is hard to capture how much a device leaks in bits, per function call, as a number less than the key size, due to the way a device leaks. It can be argued that the most a device can leak is the number of bits in the secret key because the key is the only unknown information within a device. However, the leakage captured from a device does not directly correspond to this view of leakage. A power trace gives a reading per time point (determined by the sampling frequency
of the oscilloscope) for the entirety of the device execution. This leads to a trace of several megabytes in size while the key tends to be only a few hundred bits. The size of the trace clearly exceeds the bounds and thus a method for capturing the amount of ‘useful’ information must be found. For example, any leakage on program counters or known constants will not give any information to the adversary. There has been work within the area of leakage detection, where given a set of traces, the aim is to try and produce a metric of if the traces contain any leakage about the secret key which can be exploited [44, 84, 146]. The downside of leakage detection is that while it states if a device leaks or not, it does not quantify how much a device leaks. Therefore we will not discuss the leakage detection literature in any more detail here.

By solving the key rank problem exactly, which is of independent interest, we are able to begin to try and bound the leakage of a device per call and relate this back to the theoretical model. Informally speaking, the key rank problem states; given the results of a side channel attack in the form of a distinguishing score vector and the value of the known key, if the keys are ordered by likelihood, what position is the known key in? Using an exact key ranking algorithm we are able to obtain information about how much (in terms of bits) a device leaks per call by using key rank repeatedly after performing a DPA style attack with an increasing number of traces. The chapter is concluded with a discussion of how measuring the amount of leakage, in the described manner, relates to the theoretical bounds given in the leakage resilient papers.

The given metric only captures information about key leakage and does not capture any information about leakage on tags or encryptions etc (and therefore cannot predict if leakage can forge a tag, for example). However, this is the first work that can give some information about the number of bits revealed about the key. Information leakage about the key relates back to the CKR-UL problem from Ch 5. The CKR-UL problem states that it is hard to recover the key from only leakage on the key update mechanism. If a scheme is not secure against CKR-UL then it cannot meet any other leakage resilient security notion, as the adversary can learn the key. The key rank algorithm will give information about bits leaked for the CKR-UL game and not other more general security notions but this is the first time an actual bound has been able to be calculated, and therefore is a step towards the goal of measuring leakage.
6.1. Notation and Preliminary Definitions

We use a bold font to indicate a vector, indices in superscript to refer to column vectors and indices in subscript to denote row vectors. If two indices $i,j$ are given then they refer to row $i$ and column $j$. When column vectors are required to be typeset in line, we write them as row vectors and use a superscript $T$ to denote the transpose e.g. $k = (...)^T$.

We partition an key guess $k$ into $m$ chunks $k = (k_0, \ldots, k_{m-1})$, each of which can take one of $n$ possible values. Our algorithm does not require that each key chunk is the same size but it helps to simplify the explanation that follows. After exploiting some leakage $l$, we assume that the side channel distinguisher outputs a matrix of probabilities $d$, such that $d_{i,j} = \Pr[k_j = i]$. This will sometimes be referred to as the distinguishing vector. There has been several works showing how to convert the distinguishing vector into this form, e.g. $[27, 126, 164]$. We use $sk_j$ to denote the value of the $j$th chunk of the secret key $sk$. The posterior probability $D$ of the secret key is then

$$D = \prod_{j=0}^{m-1} d_{sk_j,j}.$$ 

This will later be mapped to (integer) weights (using a function $\text{MapToWeight}$) and the weight of the secret key will be denoted $W$ where $W$ is defined as $W = \sum_{j=0}^{m-1} \text{MapToWeight}(d_{sk_j,j})$.

With respect to AES: Kocher et al.’s attack [103] consists of using a t-test statistic as a distinguisher to compute scores for the values of each 8-bit chunk of the 128-bit key. See Fig. 6.1 for a visual example. Here, AES-128 has $m = 16$ chunks, each containing $n = 256$ values, with associated distinguishing scores as derived via a t-test statistic. In the graphical illustration, the

<table>
<thead>
<tr>
<th></th>
<th>$d^0$</th>
<th>$d^1$</th>
<th>$d^2$</th>
<th>$d^{m-1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.01</td>
<td>0.01</td>
<td>0.40</td>
<td>0.02</td>
</tr>
<tr>
<td>1</td>
<td>0.03</td>
<td>0.13</td>
<td>0.05</td>
<td>0.03</td>
</tr>
<tr>
<td>2</td>
<td>0.41</td>
<td>0.11</td>
<td>0.20</td>
<td>0.31</td>
</tr>
<tr>
<td>3</td>
<td>0.38</td>
<td>0.27</td>
<td>0.02</td>
<td>0.33</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\ddots$</td>
<td>$\vdots$</td>
<td>$\ddots$</td>
</tr>
<tr>
<td>$n-2$</td>
<td>0.09</td>
<td>0.01</td>
<td>0.30</td>
<td>0.12</td>
</tr>
<tr>
<td>$n-1$</td>
<td>0.07</td>
<td>0.02</td>
<td>0.01</td>
<td>0.04</td>
</tr>
</tbody>
</table>

Figure 6.1.: Example probabilities for $m$ key chunks. Each chunk can take values from 0 to $n-1$. The values that correspond to the (hypothetical) secret key are highlighted in grey.
secret key values are marked out in grey.

**Definition 43 (Key Rank).** Given an \( n \times m \) matrix of probabilities \( d \) and posterior probability \( D \) of the secret key \( sk \), the rank of \( sk \) is defined as the number of keys with a posterior probability greater than \( D \). Formally:

\[
\text{rank}_{sk}(d) = | \{ k = (k_0, \ldots, k_{m-1}) | \prod_{j=0}^{m-1} d_{k_j,j} > \prod_{j=0}^{m-1} d_{sk_j,j} \} |
\]

An equivalent description is: Given an \( n \times m \) matrix of probabilities \( d \) and posterior probability \( D \) of the secret key \( sk \), at what position is \( sk \) in an ordered (decreasing) list sorted by the probabilities of the keys? These two notions are equivalent because any key that occurred higher in the list than \( sk \) would have a probability more likely than \( D \).

Without loss of generality it can be assumed that we are counting the keys with a score strictly larger than \( D \). This means that if several keys have the same score the target key will be ranked top. This is the more desirable metric as it represents the worst case, in that, if an adversary follows an optimal strategy, it is the earliest that they can find the key. For our particular algorithm it is trivial to convert such that for equally ranked keys the target key is returned as the lowest rank. This can be achieved by replacing some strict inequalities with inequalities. Having the target key ranked last among equally likely keys will be useful for considering theoretical leakage. If all keys are equally likely, ranking the target key as first implies it has completely leaked out, which is not true when considering leakage. Other metrics, such as average case, can be calculated using the upper and lower ranks given by the algorithm.

**Definition 44 (Key Enumeration).** Given an \( n \times m \) matrix of probabilities \( d \) and an integer \( B \), output the \( B \) keys with the highest posterior probabilities.

Note that the definition only asks for the \( B \) most likely key, not that they are returned in order. When \( B \) is small the problem of returning the keys in order is equivalent because it is possible to find the \( B \) most likely keys, sort them and return them in order. However, when \( B \) is large enough, the keys cannot be stored but have to be tested and then discarded. In this situation, the version which requires the \( B \) keys to be returned in order is strictly a harder problem. From this
point forward we will focus on key enumeration (without strict ordering). When the $B$ keys are
required to be returned in order it is referred to as the optimal key enumeration problem.

6.2. Previous Work

In this section we give the previous work for both the key rank and key enumeration problem.
Once the previous work has been given we reflect on the state of the area and discuss the issues
that have arisen and the direction we believe that the area should take to solve the key rank and
key enumeration problems.

6.2.1. Key Rank

The trivial way to solve the key rank problem is to take the vector of probabilities and calculate
the posterior probability for each possible key. Once the probability of each key is known, the
keys can then be sorted by their probability and the rank of a particular key is then just its
position within the sorted list. While this is doable for a small number of keys, in the case of
AES-128 (which is the commonly used example in the literature) there are $2^{128}$ keys, at which
point this method is no longer computationally feasible. Hence, a method to calculate the rank
of the key is required without having to calculate all possible key scores.

Veyrat-Charvillon et al. [164]

Veyrat-Charvillon et al. [164] were the first to demonstrate an algorithm to rank a known key
without first having to fully enumerate the keys. They show that the key space can be represented
as a multidimensional space where each dimension corresponds to a key chunk, which is sorted
by its likelihood. This space can then be divided into two; those keys with a higher posterior
probability than the target key and those key with a lower one. Using the property that the
‘frontier’ between these two sets is convex (since choosing a less likely value for a key chunk
can only decrease the total probability), they are able to prune both sets until the rank of the key
is estimated to within an accuracy of ten bits. Where ten bits of accuracy is defined as; if the
algorithm returns that the key has rank position $2^x$, then the actual rank, as determined by the
probability distribution, is within the range $2^{x-10}$ and $2^{x+10}$.

**Berstein et al. [27]**

Berstein et al. [27] propose two key ranking algorithms. The first expands on the work of Veyrat-Charvilion et al. [164] by adding a post processing phase which tightens the accuracy from 10 bits to 5 bits, using statistical sampling. The second algorithm given in the paper uses techniques similar to those used to count all $y$-smooth numbers less than $x$ to construct a rank estimating algorithm. Unlike previous algorithms, this algorithm has an accuracy parameter which allows the user to increase the runtime to make the bounds as tight as required.

**Glowacz et al. [78]**

Glowacz et al. [78] create a key rank algorithm using novel techniques. The first step is to turn the probabilities into additive scores by taking their logarithm. Then using the property that the set $S_1 + S_2 := \{x_1 + x_2 | x_1 \in S_1, x_2 \in S_2\}$ can be approximated by the convolution of histograms, they are able to construct an extremely efficient key rank algorithm. The algorithm is able to estimate the key to within a single bit of precision. Due to the accuracy provided by their algorithm and the efficiency, we will use this algorithm as a benchmark on which to compare our scheme.

**Duc et al. [59]**

Duc et al. [59] create a key rank algorithm inspired by the techniques of Glowacz et al. [78]. They ‘merge’ the data in one chunk at a time; similar to how the histograms were convoluted one chunk at a time. After each merge the resulting information is ‘downsampled’ to the closest of a predetermined set of discrete values. This can be seen as similar to putting the values in histogram bins. However, unlike the work of Glowacz et al. [78], the downsampling does not just happen at the beginning of the procedure but after each merge has taken place.
6.2.2. Key Enumeration

In a similar manner to key rank, key enumeration can be solved by calculating the probabilities of all the keys, sorting the list and returning the $B$ most likely items. However, as previously stated, this is not computationally feasible and thus another method is required.

It is worth noting that while it is difficult for an arbitrary value of $B$, when $B = 1$ the problem is easy to solve because taking the most likely value for each subkey gives the most likely key. Frequently within the literature $B = 1$, such as, when a work states that an attack is successful when the most likely key returned by the attack is the correct key [110].

**Veyrat-Charvillon et al. [163]**

Veyrat-Charvillon et al. [163] provide a deterministic algorithm to enumerate keys using a divide-and-conquer approach. They use a tree-based recursion on the number of key chunks (each of which is sorted by likelihood) starting with two, doubling until all sixteen key chunks are considered. By keeping track of the frontier set (similarities to the frontier of Veyrat-Charvillon et al. [164] can be drawn) they are able to efficiently enumerate keys by choosing the most likely.

**Ye et al. [169]**

Ye et al. [169] define and construct a key space finding algorithm. A key space finding algorithm is given the distinguishing score vector as input and produces two outputs. The first is the minimum complexity to ensure a desired success probability to find the key, while the second is the probabilistic strategy the adversary can use to achieve this bound. Given these two outputs, it is possible to run a probabilistic key enumeration algorithm. Intuitively, the distinguisher keeps adjusting the ‘boundary’ of which keys should be enumerated until the desired success probability has been achieved, at which point the algorithm terminates.

**Bogdanov et al. [30]**

Since the publication of our work [113], Bogdanov et al. [30] have designed a score-based key enumeration algorithm (SKEA). SKEA is a deterministic algorithm which assigns integer
scores to key chunk values, based on the probabilities, from a given range (which depends on the desired precision). Working backwards from the maximum score to the minimum score it finds all keys with the given score. This is made efficient using backtracking techniques, together with pruning impossible paths, as early as possible. Similar to our scheme, it can be highly parallelised based on the number of cores there are available for it to run on; that is to say there is no inherent algorithmic restriction.

6.2.3. Reflecting on the Previous Work

All of the previous work has treated the key rank and key enumeration problems as two distinct problems and only approached solving one of them. Due to this, the methods used to solve each of the problems are disjoint and different techniques have been built upon for each problem. As a direct consequence of this, it is not clear how to extend the solution of one problem to the other. For instance, it is not apparent how to take the key enumeration algorithms and simplify them to perform key rank without enumerating until the desired key is found. Similarly, it is not immediately obvious how to add book keeping to the key rank algorithms so that enumeration of the keys may also be performed. Another advantage of having a single algorithm to perform both is that the implementation cost is greatly reduced since there is a large amount of overlap in the implementation details for both algorithms. Therefore, increasing the efficiency of one algorithm should also increase the efficiency of the other.

We, however, believe that these two problems are highly related and should be treated as such. Therefore, it should be possible to construct a key rank algorithm which, with some additional book keeping, can perform key enumeration. In this chapter we show just that; how to construct a key rank algorithm which can be extended so that it can also perform key enumeration. We then conclude the chapter by using the key rank algorithm to investigate the theoretical bound used within leakage resilient schemes.
6.3. Key Rank

6.3.1. Casting as a Knapsack Problem

We start by explaining how the key ranking problem can be expressed as a variant of the knapsack problem. The ‘standard’ knapsack problem takes in a set of \( n \) items \( \{z_i\}_{i=0}^{n-1} \) and a total weight \( W \). Each item \( z_i \) has a corresponding profit \( p_i \) and integer weight \( w_i \). A series of binary variables \( \{x_i\}_{i=0}^{n-1} \) are used such that \( x_i \) selects whether \( z_i \) is chosen from the set. The objective is to select items from \( \{z_i\}_{i=0}^{n-1} \) such that the profit is maximised, without exceeding the total weight \( W \). Formally:

\[
\text{maximise} : \sum_{i=0}^{n-1} p_i \cdot x_i \\
\text{subject to} : \sum_{i=0}^{n-1} w_i \cdot x_i \leq W
\]

It has been shown that the decisional knapsack problem (which asks if there is a solution with profit at least \( P \), for chosen value of \( P \)) is Non-deterministic Polynomial-Time (NP) complete [76], while the optimisation problem (given above) is NP hard [76]. The optimisation problem being NP hard implies that given a solution, there is (currently) no (known) way to show that it is optimal in polynomial time. There is a pseudo-polynomial time algorithm to solve the problem using techniques from dynamic programming. Pseudo-polynomial is defined as the algorithm being polynomial in the value of the input and not its size (in this case polynomial in the input size and pseudo-polynomial in \( W \)). Our technique for solving the key rank problem shares similarities with this algorithm and will also be polynomial in the value of the weight. The knapsack algorithm works by storing values \( dp[i][j] \) which corresponds to a solution only using items \( i, \ldots, n - 1 \) with weight at most \( j \). At step \( i \) the two options are to either add \( i \) to the solution (so must look at the items from \( i + 1 \) of weight at most \( W - w_i \)) or not to add \( i \) to the solution (using the solution for \( i + 1 \)). This leads to the following recurrence relation, giving an algorithm to maximise profit which is pseudo-polynomial in the weight \( W \).
The counting knapsack (#knapsack) problem is defined as the equivalent counting problem for the knapsack problem. Given a total weight \( W \) and a set of \( n \) items \( \{z_i\}_{i=0}^{n-1} \), each of which has a weight \( w_i \), how many ways are there to assign value to the binary variables \( \{x_i\}_{i=0}^{n-1} \) such that \( \sum_{i=0}^{n-1} w_i \cdot x_i \leq W \)? In this definition the profits are no longer given because the problem requires all viable solutions to be counted and not for the profit to be maximised.

The counting knapsack problem is known to be \#P complete [85]. \#P is the class of problems of the form; “Compute the number of accepting paths for a given NP machine”. Gopalan et al. [85] present a Fully Polynomial Time Approximation Scheme (FPTAS) for solving the #knapsack problem.

The #knapsack problem is closer to the key rank problem since we are trying to find how many solutions there are instead of finding a maximal solution. However, unlike the standard knapsack problem, we have classes of items, corresponding to the probability vectors \( d_i \), and we also have the restriction that only one item can be chosen from each class. The integer weight \( w_{i,j} \) can be derived from the corresponding probabilities \( w_{i,j} = \text{MapToWeight}(d_{i,j}) \) in such a way that higher probabilities lead to lower weights. Since finding the rank of a key corresponds to finding all keys with a weight less than the secret key’s weight; more likely keys need to have lower weights so fewer keys are above them, while lower ranked keys (small probabilities) have higher weights so more keys have weight less than them. We define the weight \( W \) to be the sum of the weights associated with the secret key \( W = \sum_{j=0}^{m-1} w_{sk,j} \). Since for equally ranked keys, the secret key is ranked first, \( W \) becomes a strict upper bound in the knapsack definition. The description of \( \text{MapToWeight} \) will be discussed in more detail in Sect. 6.5 but for the description of the functions it is enough to know that the function exists in abstract.

The key rank problem is suited to being represented as a multiple-choice knapsack problem, instead of a standard knapsack problem. In a multiple-choice knapsack problem the items are subdivided into \( m \) classes \( \{N_j\}_{j=0}^{m-1} \) and exactly 1 item must be chosen from each class. Formally:

\[
\begin{align*}
\text{dp}[i][j] = \max \left( \begin{array}{c}
dp[i+1][j-w_i] + p_i \ (j \geq w_i) \\
dp[i+1][j]
\end{array} \right)
\end{align*}
\]
maximise :  \[ \sum_{j=0}^{m-1} \sum_{i \in N_j} p_{i,j} \cdot x_{i,j} \]

subject to :  \[ \sum_{j=0}^{m-1} \sum_{i \in N_j} w_{i,j} \cdot x_{i,j} \leq W \]
\[ \sum_{i \in N_j} x_{i,j} = 1 \quad \forall 0 \leq j < m \]

The multiple-choice knapsack problem is NP hard since it has the knapsack problem as a special case [135]. For each item in the knapsack problem add it into a set with a 0 cost, 0 profit item. Therefore, choosing an item from the set in the multiple-choice knapsack problem directly corresponds to choosing, or not choosing, an item in the original knapsack problem. To choose an item from the original knapsack, the corresponding item is chosen from the set. While, if an item is not selected from the original knapsack, the corresponding zero weight item is selected.

If the counting variation of this problem is considered, a notion of profit is no longer required because the problem is counting valid solutions and not trying to maximise over all solutions. The key rank problem can be mapped to this problem. The requirement that the sum of the weights is less than the total weight \( W \) ensures only solutions with a lower weight than \( W \) are considered and, therefore, the corresponding key has a rank higher than the secret key. The final constraint ensures that only one item is chosen from each class. This corresponds to making sure that a single value is chosen for each key chunk and hence only valid keys are considered. Thus, the counting multiple-choice knapsack problem can be used to solve the key rank problem.

For ease of illustration it will be convenient to express our multiple-choice knapsack problem as a multi-dimensional knapsack problem. In the multi-dimensional knapsack problem each item \( z_i \) has a vector of weights \( (w_1, \ldots, w_l) \) and the items chosen must maximise the profit while meeting the weight constraints for each component of the vector. Formally:
maximise : \[ \sum_{i=0}^{n-1} p_i \cdot x_i \]
subject to : \[ \sum_{i=0}^{n-1} w_{i,j} \cdot x_i \leq W_j \forall 0 \leq j \leq l \]

The multi-dimensional knapsack problem is NP-hard, since the knapsack problem is the case where each item only has a single weight.

For the key rank algorithm the multi-dimensional knapsack will be defined as follows; each key item \( k_{i,j} \) will have an associated weight vector \( w_{i,j} = (w_{i,j}, 0, \ldots, 0, 1, 0, \ldots, 0) \) where the 1 occurs in position \( j + 1 \). The weight constraints can also be expressed as a vector \( W = (W, 2, \ldots, 2) \) where all vectors are of length \( m + 1 \). The key rank problem can then be defined as counting all the solutions to the following constraint, where addition and constraints within vectors are defined component-wise.

\[
\sum_{j=0}^{m-1} \sum_{i=0}^{n-1} w_{i,j} \cdot x_{i,j} < W
\]

The constraint \( W \) ensures that all keys that are counted have weight strictly lower than \( W \) and thus are ranked higher than our secret key. If the weight vector has a 1 in position \( j + 1 \) it corresponds to the fact that it represents a value for the \( j^{th} \) key chunk. Since the weight limit is two in the constraint vector \( W \), only one value may be chosen with a 1 in position \( j \) and, therefore, only one value may be chosen for the \( j^{th} \) key chunk.

**Running Example**

In this subsection we introduce the running example which will be used throughout the remainder of this chapter to help illustrate the working of our algorithms. In the example we consider a key composed of two independent subkeys, each of which can take one of three possible values (0-2). For this example, consider the following weight table:
Let the secret key \( sk \) be \((2, 1)\). This gives the key a weight of \( W = w_{2,0} + w_{1,1} = 5 \).

The weight constraint vector has the value \( W \) in the first position, to restrict keys to have a weight less than our secret key. The remaining values are set to 2 to enforce that only one value can be chosen per subkey. The resulting weight vector, for our two subkey example, is then \( W = (5, 2, 2) \). The weight vectors \( w_{i,j} \) corresponding to key value \( k_{i,j} \) have the weight \( w_{i,j} \) in the first position and a 1 in position \( j + 1 \) to correspond to this being a value for the \( j^{th} \) subkey, the remaining positions are set to zero. This gives the following weight vectors:

\[
\begin{align*}
w_{0,0} &= (0, 1, 0), \quad w_{1,0} = (1, 1, 0), \quad w_{2,0} = (3, 1, 0) \\
w_{0,1} &= (0, 0, 1), \quad w_{1,1} = (2, 0, 1), \quad w_{2,1} = (3, 0, 1)
\end{align*}
\]

Given these constraints, the solutions to the knapsack are:

\[
(k_{0,0}, k_{0,1}), (k_{0,0}, k_{1,1}), (k_{0,0}, k_{2,1}) \\
(k_{1,0}, k_{0,1}), (k_{1,0}, k_{1,1}), (k_{1,0}, k_{2,1}) \\
(k_{2,0}, k_{0,1})
\]

It is important to note that the solution does not include the secret key, as it is looking for weights strictly less than the target weight. However, this corresponds with the definition of key rank, giving our key in the above example a rank of 7.

Given the knapsack instance at this stage, it is possible to solve the knapsack using known techniques. A knapsack instance can only be solved exactly using a pseudo polynomial dynamic programming algorithm; it is polynomial in the weight of the knapsack. However, it becomes more complex to solve these more generalised knapsack problems using this type of solution.
the following section we demonstrate how to solve the problem for our particular instance which arises from the key rank problem.

6.3.2. Key Rank Graph

In this section we show how our particular multi-dimensional knapsack problem can be represented as a graph, where the number of paths from a designated start vertex to a designated accept vertex corresponds to the number of solutions for our knapsack problem. While, in general, $s$-$t$ path counting is hard (it is #P complete [139]) we show that, for our particular graph, the number of paths can be counted efficiently, leading to an efficient algorithm for ranking a key.

Recall that our multi-dimensional knapsack problem has $n \cdot m$ items and each item has a weight vector. Since all valid solutions have a weight, in the first component, less than $W$, a way to encode the weight of a partial solution while it is less than $W$ is required (as soon as the weight is greater than $W$ it can be rejected). Since this information is required for each step in the computation, $n \cdot m \cdot W + 2$ vertices in our graph are required; $W$ vertices for each element in our knapsack to keep track of partial weights plus an accept and reject state. Since the other weights in $W$ encode the fact that only one element from each key chunk can be chosen, we can construct the graph in such a manner to enforce this, instead of using a similar counting trick to the one above (if the weights were higher, the above technique would have to be used). The vertices corresponding to item $k_{i,j}$ are denoted as $V_{i,j}^w$ where the $w$ is used to denote the partial weight. The key rank graph contains a start vertex $S = V_{0,0}^0$, an accept vertex $A$ and a reject vertex $R$. The edges for the graph are then constructed as follows:

1. $(V_{i,j}^w, V_{i+1,j}^w) \forall 0 \leq i < n - 1, 0 \leq j \leq m - 1, 0 \leq w \leq W - 1$
2. $(V_{n-1,j}^w, R) 0 \leq j \leq m - 1, 0 \leq w \leq W - 1$
3. $(V_{i,j}^w, V_{0,j+1}^{w+w_{i,j}}) \forall 0 \leq i \leq n - 1, 0 \leq j < m - 1, w + w_{i,j} < W$
4. $(V_{i,m-1}^w, A) \forall 0 \leq i \leq n - 1, w + w_{i,j} < W$
5. $(V_{i,j}^w, R) \forall 0 \leq i \leq n - 1, 0 \leq j < m - 1, w + w_{i,j} \geq W$

217
The first rule for creating edges corresponds to the rule where a value $i$ is not chosen for key chunk $j$ and, therefore, the next possible item $i + 1$ is considered for the same key chunk and the weight $w$ does not change. If this was the last element in key chunk $j$, rule two is applied and the edge goes to the reject vertex because this represents no value being assigned to key chunk $j$ (which would not produce a valid key). The third rule corresponds when item $i$ is chosen to be the item for key chunk $j$ (and is a valid choice); the partial weight must be incremented by $w_{i,j}$ as this is the chosen item’s weight and then, since the single item has been chosen for this key chunk, the edge must point to the beginning of the next key chunk (assuming this was not the last key chunk). Rule four is the equivalent rule to rule three, for when the key chunk being considered is the last key chunk, if it is a valid choice, it can no longer go to the next key chunk and so goes to the accept vertex. This corresponds to constructing a complete key with total weight less than $W$. The final rule corresponds to the case where item $i$ is chosen to be the value for key chunk $j$, while it is a valid choice, because the weight is higher than $W$, it corresponds to a key with lower rank than the secret key and thus it should not be counted (and therefore goes to reject).

Running Example

In this subsection we continue with our running example (recapped in the top left of Fig. 6.2) and, using the data given, we construct the corresponding graph. Since our secret key has target weight 5, we have a graph of width 5. The columns of the graph will be used to track the partial weight of the keys, representing weights 0 to 4. As the graph has 2 key chunks, each of which can take one of three possible values, the resulting graph has 6 rows. Given these vertices, plus the accept and reject vertices, gives a graph containing 32 vertices, where $S = V_{0,0}^0$ (as seen in the top right of Fig. 6.2).

The middle left diagram of Fig. 6.2 shows the first two edges to be added. The first edge goes from $V_{0,0}^0$ to $V_{1,0}^0$. This corresponds to not selecting value 0 for the 0th key chunk and therefore value 1 must be considered. This utilises rule one. The second edge added is between $V_{0,0}^0$ and $V_{0,1}^0$ and corresponds to choosing the value 0 in key chunk 0. The edge ‘jumps’ to value 0 in key chunk 1 to enforce that no two values can be chosen for the same key chunk.
Figure 6.2.: The construction of the graph for the small example instance provided [113]
chunk. The weight is incremented by 0 because this is the weight of value 0 for key chunk 0. This utilises rule three.

The middle right diagram of Fig 6.2 adds four more edges to the graph. Two of them \( V_{1,0}^0 \) to \( V_{2,0}^0 \) and \( V_{0,1}^0 \) to \( V_{1,1}^0 \) were added using rule one. These correspond to not choosing the given value for that key chunk and, thus, moving onto the next value. An edge is added between \( V_{1,0}^0 \) and \( V_{0,1}^{0+1} \) to correspond to the value 1 being chosen for key chunk 1. As when 0 was chosen it points to the row for value 0 in key chunk 1, not the current value, so that all values in the next key chunk can be considered. The weight is incremented by 1, since value 1 in key chunk 0 has weight 1, making the partial weight so far 1. This is carried out using rule three. An edge is added from \( V_{0,1}^0 \) to \( A \) using rule four. This corresponds to choosing value 0 from chunk 1 which increases the partial weight from 0 by 0 and, thus, giving a total weight less than 5. This implies it represents a valid key that will have rank higher than our secret key.

The process continues using the provided rules until all edges have been created. The last item in each chunk will have a left child that points to reject (as there are no further items in the chunk to select) but these have been omitted from the example diagram for the sake of clarity. These are created using rule two. All the greyed out vertices also have their children calculated. However, as they do not alter the path count, we have excluded them from the example figures to aid clarity. Each path from the start vertex \( S \) to the accept vertex \( A \) corresponds to a key with lower weight than our secret key’s weight \( W \). Therefore, counting the paths between \( S \) and \( A \) will correspond to the rank of the key. While, in general, path counting is hard [139], we explain how our graph structure, having, at most, two outgoing edges per vertex, lends itself to constructing an efficient path counting algorithm.

Given this graph; there are 7 paths from \( S \) to \( A \). This is exactly the rank of the key in our example which was calculated previously.

### 6.3.3. Counting Valid Paths

We have shown how to represent our multi-dimensional knapsack instance as a graph and argued that counting paths between the vertex \( S \) and the vertex \( A \) is equivalent to computing the rank of the key. In this section we show how to calculate the number of paths between \( S \) and \( A \).
efficiently. While, in general, $s$-$t$ path counting is a hard problem, since all our vertices (with the exception of $A$ and $R$ which are sinks) have out degree two, it is possible to construct an efficient counting algorithm using dynamic programming for our specific case, which is given here.

From this point forward we will assume that our graph is topologically sorted. A list of vertices is said to be topologically sorted if for every edge $(U,V)$, $U$ comes before $V$ in the ordering. It is possible to topologically sort our graph because, based on the rules, the resulting graph will be a Directed Acyclic Graph (DAG). While, for the key rank algorithm, how the graph is topologically sorted does not matter (all that matters is that it is sorted - the particular sorting does not change the time complexity), for key enumeration, how the graph is topologically sorted affects the efficiency and memory requirements of the algorithm and, thus, will be discussed in more detail within the relevant section (Sect. 6.4). We will label the start vertex $S$ as 1 and the final vertex $A$. Since there are $n \cdot m \cdot W + 2$ vertices; $A = n \cdot m \cdot W + 2$ and $R = A - 1$. The order of the other vertices does not matter for this algorithm, only that they are sorted.

We also assume two constant time functions $LC(\cdot)$ and $RC(\cdot)$ (which will depend on the topological sorting) which return the index to the left child and right child of a vertex respectively. When relating this to the key rank problem, the left child function corresponds to the decision of not choosing the item for the key chunk, while the right child function represents choosing the item for the given key chunk. One way to make these functions constant time is to store them as a look up table. However, due to the structure of our graph, for various topological sortings, these values can be calculated explicitly in constant time. Not only does this make computing the children efficient, it means the graph does not explicitly have to be built to run the algorithm. This will be discussed more in the key enumeration section where the topological sorting is more important.

Let $PC$ be a vector of length $A$ such that $PC[c]$ stores the number of paths from $c$ to $A$, then we define the following recurrence relation.
Given this recurrence relation, the number of paths between \( S \) and \( A \) is simply the value of \( PC[1] \). Assuming that \( LC \) and \( RC \) are globally accessible functions, we can use the recurrence relation to construct the following algorithm (Fig. 6.3) for calculating the number of paths between \( S \) and \( A \), which corresponds to the key rank of the secret key.

\[
PC[c] = \begin{cases} 
1 & \text{if } c = A \\
0 & \text{if } c = A - 1 \\
PC[LC(c)] + PC[RC(c)] & \text{otherwise}
\end{cases}
\]

\[
PC[A] \leftarrow 1 \\
PC[R] \leftarrow 0 \\
\text{for } c = A - 2 \text{ to } 1 \text{ do} \\
\quad PC[c] \leftarrow PC[LC(c)] + PC[RC(c)] \\
\text{end for} \\
\text{Return } PC[1]
\]

Figure 6.3.: The Key Rank Algorithm

Correctness

There are two base cases for the algorithm. The first base case is \( PC[A] = 1 \). This holds because, there is only one way from a vertex to itself (in a DAG) – the path involving no edges.

The second base case is \( PC[R] = 0 \). By design of the graph, there is no way to move from the reject vertex to the accept vertex (both are sinks) and, thus, there are zero paths between \( R \) and \( A \). For an arbitrary \( 1 \leq c \leq A - 2 \), there are two outgoing edges from this vertex; the edge to left child (therefore counting the number of paths from the left child to \( A \)) or the edge to the right child (therefore counting the number of paths from the right child to \( A \)). This gives the relation \( PC[c] = PC[LC(c)] + PC[RC(c)] \) as required. Since all the vertices are iterated over in reverse (from \( A \) to 1) and the graph is topologically sorted, whenever a vertex is being operated on, the path count for its two children will already have been calculated (otherwise the graph would not have been topologically sorted) and, thus, the values required have already been computed.
**Time Complexity**

The time complexity of the algorithm depends on the number of vertices and the size of the integers stored in $PC$. Since the graph is topologically sorted, each path count from $c$ to $A$ only has to be calculated once and the loop performs $O(A)$ iterations. The integers stored can be, at most, double the value stored by their children (when $PC[LC(c)] = PC[RC(c)]$). As the loop has $O(A)$ iterations, the integers can be, at most, $O(2^A)$ (since the integers with a value of 1 or 0 at the base case). Therefore, addition of the two integers takes $O(A)$ time (since an integer of size $2^b$ requires a $b$ bit representation) and the whole algorithm has a time complexity of $O(A^2)$. Since for our particular case, $A = O(n \cdot m \cdot W)$, the algorithm has a runtime of $O(n^2 \cdot m^2 \cdot W^2)$.

However, for our particular graph, we can construct a tighter bound. While each element in the array has to be touched once to fill in the value, there are, at most, $n^m$ keys and thus the values stored in $PC$ are $O(n^m)$ instead of $O(2^{nm+W})$. Therefore addition takes $O(m \cdot \log n)$ time giving the overall time complexity for the key rank algorithm to be $O(m^2 \cdot n \cdot W \cdot \log n)$.

It is worth noting that the depth of the key does not factor in to the time complexity of the key rank algorithm. The following example will help to clarify why this is indeed the case. Consider the secret key for AES-128 which has 1’s for all its distinguishing scores. This gives $W = 16$ and $65538 = 16 \cdot 16 \cdot 256 + 2$ vertices in the graph (for an 8 bit attack). If all other key values in the distinguishing vector have value zero then the secret key will have rank $2^{128} - 1$ because all other keys will have lower weight. On the other hand, if all other key values have weight two then our target key will be rank 0 because all other key have a higher weight. Since none of the weights of the other key values change the size of the graph or $W$, the key rank will not affect the runtime. The weights of the other key values do not affect the size of the graph but do affect the structure of the graph, as the other key values’ weight appear in the definition of where edges are placed within the graph.

The algorithm runs in $O(W)$ when a particular algorithm and implementation have been chosen (fixing $m$ and $n$). For our particular case of AES-128 the values of $m$ and $n$ are fixed to 16 and 256 respectively. The results within the implementation section support this (Sect. 6.5). Note that $m$ and $n$ are fixed for the implementation and algorithm, not just for the algorithm.
Figure 6.4.: The array generated when running the key rank algorithm for the running example alone; it would be possible to consider a 32 bit implementation of AES-128 where we would have $m = 4$ and $n = 2^{32}$. However, we leave investigating this case as future work, as while the number of keys has not changed, with this implementation the size of the graph has grown greatly. The graph has $n \cdot m \cdot W$ vertices, so for the case of AES-128 as an 8 bit implementation there are $2^{12} \cdot W + 2$ vertices, while in a 32 bit implementation there would be $2^{36} \cdot W + 2$ vertices.

**Running Example**

At this stage we can complete our running example for key rank by showing how the algorithm works on our example. Instead of creating the graph as shown in Fig. 6.2, the graph is implicitly represented as an array. The children are calculated using the left child and right child functions (given in Table 6.1 for different topological sortings of the graph). Iterating through the array backwards, using Algorithm 6.3, results in the array given in Fig. 6.3.3

The last element in the array corresponds to the accept vertex and, therefore, has value 1, while the penultimate element in the array corresponds to the reject vertex and is set to 0. From this point onwards, for each vertex, the children are calculated using the functions given and the results added together to give the result. For example, vertex 1 has children 6 and 16 (using the wide sort) which have values 4 and 3 respectively, giving the required total of 7. Working backwards through the array, results in the start vertex (the first element in the array) being assigned the value seven. This corresponds to the rank of the key. This completes our running example of how to calculate the rank of a key given the integer weights.


\[ KL[A] \leftarrow \emptyset \]

\textbf{for} \( c = A \) \textbf{to} 1 \textbf{do}

\[ KL[c] \leftarrow (\text{value}(c), KL[RC(c)]) \cup KL[LC(c)] \]

\textbf{end for}

\textbf{return} \( KL[1] \)

Figure 6.5.: The key enumeration algorithm

\section*{6.4. Key Enumeration}

We are able to modify the key rank algorithm given previously, such that, with standard bookkeeping adjustments, we are able to list all valid paths as opposed to just counting them. The paths correspond directly to the keys, enabling us to list the required keys. This technique is known within the algorithmic community and is similar to the technique used to make shortest path algorithms return the shortest path and not just the weight of the shortest path. The algorithm is given in Fig. 6.5 and requires an additional (constant time) function \text{value} which, given an index, returns the value at that index (it can be seen that this is fairly straightforward to do). We write \((a, \{x_c\})\), to denote \(\{(a, x_c)\}_c\). That is, when an element is concatenated with a set, the item \(a\) is concatenated with every item in the set to form a new set.

The algorithm works as follows; every time a right child is taken (children can be computed using the \(RC\) and \(LC\) functions as with the key rank algorithm) the corresponding value for this key chunk item is chosen and, therefore, needs to be appended to the partial key list (for all options in the key list). If the left child is taken, this corresponds to not choosing the value for this key chunk and, thus, nothing is appended to the partial key list. The correctness of this algorithm follows from the correctness of the key rank algorithm combined with the fact that the book keeping techniques keep track of what values for each key chunk have been selected.

The algorithm given in Fig. 6.5 takes in a weight \(W\) and enumerates all keys with a weight smaller than the total weight. Hence, if the weight \(W\) is associated with a key \(sk\), all keys with a higher rank than \(sk\) can be enumerated. Using this algorithm, it is then possible to create a key enumeration algorithm. If the goal is to enumerate \(B\) keys then a weight can be chosen (corresponding to a key guess’ weight, from the distinguishing vector) and the key rank algorithm can then be run using this weight before comparing the result to \(B\). This process can
be repeated until the rank of the chosen as rank close enough to \( B \). This weight then can be used to enumerate the keys. Binary search can be performed over the weight vector to select a suitable key candidate. We found that simple incrementation, of the element chosen in single columns, of the weight table, worked suitably well for our purposes.

While the algorithm returns the top ranked \( B \) keys, the keys are not guaranteed to be returned in rank order. The order they are returned in is dependant on the order that the values are stored within each key chunk. For example, if the key chunks are stored in numerical order (as with all of our examples) the resulting top \( B \) keys will be returned in numerical order. Therefore, if the goal is to meet the stronger definition in which the top \( B \) keys are returned in order, the items in the individual chunks must be stored in an order to reflect this and the order the key chunks are checked may also become important. This may make the value function more complex than just simply being the position of an item within the key chunk. Storing the partial keys in a priority queue, using the current weights as a key, would enable the key enumeration algorithm to be optimal, at the expense of the slow down associated with the priority queue. We will not discuss this in more detail here, as it does not change the other design decisions being discussed.

In the rest of the section we discuss the runtime and memory complexity of the key enumeration algorithm. While the runtime (and memory) must be lower bounded by the number of keys to be enumerated, we show that there are different strategies to reduce the memory requirement. We conclude the section with a simple observation which allows key enumeration to become highly parallel with minimal overhead (there is no communication overhead just the set up and finalisation of the threads).

### 6.4.1. Time Complexity

As with key rank, we give two time complexities. We start with the worse case analysis for a general graph before giving a tighter bound for when our particular graph is used.

For a general graph, the key enumeration algorithm will be exponential in the number of vertices. Since each vertex has two children there are, at most, \( O(2^A) \) possible paths (since the graph is a directed acyclic graph) and as each vertex is touched once; the resulting time complexity is \( O(A \cdot 2^A) \).
if \((n \cdot W) \cdot (c \mod (n \cdot W)) \leq W\) then
  return \(R\)
else
  return \(c + W\)
end if

\[w \leftarrow c \mod W\]
\[i \leftarrow (c - w) \mod (n \cdot W)\]
\[j \leftarrow c \mod n\]

if \(w + w_{i,j} \geq W\) then
  return \(R\)
else if \(j \neq m - 1\) then
  return \((j + 1) \cdot n \cdot W + w + w_{i,j}\)
else
  return \(A\)
end if

i \leftarrow c \mod n
if \(i = n - 1\) then
  return \(R\)
else
  return \(c + 1\)
end if

\[w \leftarrow c \mod n \cdot W\]
\[j \leftarrow c - w \mod (j + 1) \cdot n \cdot W\]

Table 6.1.: Pseudo code for generating the index of each child of a vertex

However, for our particular graph, a time complexity of \(O(m^2 \cdot n \cdot W \cdot B \cdot \log n)\) can be achieved, where \(W\) was chosen such that the weight enumerates approximately \(B\) keys. To achieve this time complexity only \(B\) partial key candidates are stored (else only the previous time complexity is achieved). However, it is possible to keep track of the \(B\) most likely partial keys, so that all possible partial keys are not stored. The time complexity then occurs because all \(m \cdot n \cdot W\) nodes in the graph are visited once, where the candidate value is appended to all possible partial keys (at most \(B\)). Each key stored has length \(O(m \cdot \log n)\) (because there are, at most, \(n^m\) keys they need \(O(m \cdot \log n)\) bits to uniquely represent the keys), giving the desired time complexity.
Figure 6.6.: Different options for topological sorting for our previous example. Note that the deepest vertex in each chunk will be guaranteed to have a left child leading to R; for clarity these paths are omitted [113].
6.4.2. Memory Efficiency

It was mentioned previously, for key rank, that how the graph is sorted did not play a role in the algorithms efficiency. However, for key enumeration this is no longer the case, and while the method for topological sorting does not effect the runtime, it greatly affects the ability to parallelise the algorithm and the amount of memory required to run the algorithm. The key rank algorithm only needs to store a single integer per node in the graph, whereas key enumeration will need to store a list of partial keys per graph vertex and thus memory efficiency will be a lot more important. While there are a wide variety of (generalised) topological sorting algorithms in the literature [98, 160], we were able to avoid the explicit sorting of our graph due to the structure of our graph. We show that our graph can be sorted implicitly based on how we initially number the vertices, and how the children are calculated within the left and right children functions.

In Fig. 6.6 we give three possible options for how the graph from our running example can be topologically sorted. Table 6.1 gives the pseudocode for the left and right children functions for each of the three sorting methods. The functions take in a vertex \( c \) and return the child of this vertex. The functions assume everything is 0 indexed but it is straightforward to convert if required. The functions also have access to \( n, m, W \) and the table of integer weights. We will now investigate the three sorting methods to see which is the most desirable by evaluating their pros and cons. We complete the discussion of memory efficiency with how the keys can be stored within each vertex in a more memory efficient manner. Essentially, the decision is a trade off between memory usage and the ability to parallelise. Since we are able to make other aspects of the algorithm parallel (see discussion that follows shortly) we chose to use wide sort since it is the most memory efficient.

**Wide Sort**

For the wide sort, the vertices are numbered one chunk at a time, one item at a time, along the weight in increasing order (see Fig. 6.6(a)). Formally, given a chunk, item and weight \((x, y, z)\) the vertex will be labelled with index \( c \) where \( c = x \cdot W \cdot n + y \cdot W + z \). This is a valid topological sorting of our graph, since for any given vertex its children will either be in the next chunk down (right child) or the next item down (left child) and both of these will have a higher
index, maintaining the ordering requirement.

The advantage of this sorting is that when either the key rank or key enumeration algorithm is performed the left and right children will only be either in the first row of the next chunk or the next item in the same chunk and thus, instead of storing the values calculated for the entire graph so far, only these two rows need to be stored. This gives the algorithm a memory requirement of $O(W)$ items, where depending on the algorithm, the item will either be a list of partial keys (for key enumeration) or a partial path count (for key rank). Since the total number of items stored does not change, based on the topological sorting used, we will compare topological sortings, in terms of memory efficiency (the maximum number of items that are required to be stored at any one time), using the number of items that need to be stored.

The disadvantage of the wide sort is that it is highly serial and does not support making either the key rank or key enumeration algorithms parallel.

**Long Sort**

For the long sort, the vertices are numbered one weight at a time, one chunk at a time, along the item in increasing order (see Fig. 6.6(b)). Formally, given a chunk, item and weight $(x, y, z)$, the vertex will be labelled with the index $c = z \cdot n \cdot m + x \cdot n + y$. This is a valid topological sorting because the right child will either have the same weight or a higher weight but will be in the next chunk and, thus, will have a higher index, while the left child will have the same weight but will increment the item by one and, thus, also have a higher valued index. This maintains the ordering property required and is therefore a valid topological sorting.

The disadvantage of this method is that there is no technique to optimise the memory requirements and thus the full graph must be stored. This gives a memory requirement of $O(n \cdot m \cdot W)$ items.

The advantage of the long sort is that (with careful synchronisation) it can be made parallel along the weights. This is due to the fact that the left child will always be along the same weight and will require no synchronisation as it will be computed by the same process. The right child is where synchronisation is required because if it has a larger weight it would be handled by another process. Therefore, it is required that, for a process to calculate a right child, all
processes running with higher weights have finished with the previous chunk before this process can begin. However, the synchronisation is not too complex to implement, since if all processes run at the same speed, they will always be at the same point at any given moment in time and thus finished with the previous chunk required by any other process.

**Wiggly Sort**

For the wiggly sort, the vertices are numbered one chunk at a time, one weight at a time, along the item in increasing order (see Fig. 6.6(c)). Formally, given a chunk, item and weight \((x, y, z)\) the vertex will have index \(c = x \cdot W \cdot n + z \cdot n + y\). This forms a valid topological sorting since the left child appears one item further down within the same weight and chunk that is currently being operated in, while the right child will be in the next chunk and chunks are being computed one at a time.

The memory requirements of this method are a reasonable middle ground between the previous two methods. It requires that only a single chunk is stored in memory, together with the first items of the previous chunk. This gives a memory requirement of \(O(n \cdot W)\).

The wiggly sort also finds a middle ground in terms of parallelism. Since the left child’s index is within the same weight as the parent and the right child’s index is in item zero of the previous chunk (which has already been calculated) each weight within a chunk can be run in parallel and are completely independent. Hence, as long as all processes synchronise at item zero in each chunk the rest of the algorithm key enumeration can be calculated in parallel.

**Key Storage**

The topological sorting of the graph dictates how much of the graph needs to be stored in memory at any given time. To complement this, how keys are stored within the graph can be used to reduce the memory per node within the graph.

Currently, the partial keys are stored in their entirety, at each point within the algorithm; this will become extremely inefficient. Consider the case of AES-128: there are \(2^{120}\) keys which have the first key byte as zero. Therefore, if all keys were stored, this key byte would be stored \(2^{120}\) times. A more appropriate data structure would reduce the amount of redundant data stored.
We use a tree structure as pictured in Fig. 6.7. In this tree structure each value is only stored once for each possible key prefix and the children correspond to the possible valid values for the next key chunk; therefore a path from the root to a leaf corresponds to a valid key. This key tree will now be generated by the key enumeration algorithm and then passed into a further algorithm which will reconstruct the keys for testing.

The advantage to this method of storing keys (bar the reduced memory requirement) is three-fold. Firstly, the algorithm is faster. This is not just due to the reduced memory requirement but the algorithm in fact has an improved time complexity of $O(m \cdot n \cdot W \cdot \log n + B \cdot m \cdot \log n)$. Each of the $m \cdot n \cdot W$ nodes must be touched once and at each node a single value of size $\log n$ is written. After the graph has been traversed the keys must be enumerated at cost $B \cdot m \cdot \log n$, since there are $B$ keys of size $m \cdot \log n$. Secondly, the conversion from the final key tree into a list of keys to be tested is highly (and trivially) parallelisable. Finally, the actual testing of keys (for the running AES-128 example this is checking an encryption against a known plaintext/ciphertext) can be amortised into the cost of converting the tree.

6.4.3. Parallelisation

We have touched briefly on how the choice of topological sort aids parallelism but we are able to achieve a much higher level of parallelism with a simple observation; it is possible to adjust the graph such that, instead of counting all paths with weight less than $W$, it is possible to count paths with weight less than $W_1$ but greater than $W_2$. This requires a slight modification to the $RC$ function where, if the weight is less than $W_1$ in the last chunk it would normally accept, it now only accepts if the weight is also greater than $W_2$ and rejects otherwise. To enumerate keys
between $X$ and $Y$, select $W_1$ and $W_2$ such that the corresponding keys have rank $X$ and $Y$ (this can be done using the key rank algorithm given previously), allowing enumeration between the two given points. This results in an algorithm which can enumerate disjoint batches of likely keys. Hence, the algorithm can be run multiple times where each instantiation of the algorithm generates its own batch of keys. The runtime will still only depends on $W_1$, while $W_2$ has no impact on the runtime.

All batches of keys can then be computed in parallel and require no communication except in the initial setup and the notification that the key has been found. It is thus trivial to parallelise the algorithm to the required degree. The range of the keys must be chosen carefully to avoid giving one machine all of the likely keys, while another machine has unlikely keys. By having the ranges suitably small and the machines requesting new ranges when complete, this problem can be avoided.

The other advantage of adding a lower bound weight provides is that while other algorithms in the literature can enumerate a fixed number of keys but if more keys are required at a later date the algorithm must start again from scratch. Our key enumeration algorithm, providing the deepest key enumerated is known, can calculate the weight of this key and use this to bound the algorithm and thus continue enumerating from where it stopped.

**Further Speed Optimisations**

Currently, the algorithm operates on every vertex of the graph. However, some vertices are not reachable from the start vertex (for example the greyed out vertices in the final image of Fig. 6.2) and thus the information calculated by these vertices will not be combined into the final solution. Hence, any computation time (and memory used) on these vertices is completely superfluous. By precalculating the number of valid paths from $S$ to every other vertex in the graph, any vertex with zero paths to $S$ can be ignored because any work done here will not be recombined into the final solution. The number of paths from $S$ to any vertex can be calculated using a generalisation of the key rank algorithm (it takes in the target vertex instead of assuming it is counting paths to $A$). Since the key rank algorithm requires very little memory and is extremely fast, this test speeds up the key enumeration algorithm and lowers its memory requirement. This
works because the key rank algorithm is faster than the enumeration algorithm due to its reduced workload per vertex. Hence, using the key rank algorithm to reduce the number of vertices considered in the enumeration algorithm increases efficiency of the scheme. It is possible to calculate the generalised key rank for all nodes in a single pass, increasing the performance further.

By only working with nodes that will have their result merged into the final result the algorithm guarantees that, at most, $B$ keys will be stored per node. Thus, removing the need to keep track of keys to artificially restrict each node to $B$ keys as mentioned previously. Each node which has its content merged into the final result can store at most $B$ keys because no keys are ever discarded and only $B$ keys are being returned. This provides an upperbound to the number of keys stored per node.

6.5. Implementation

In this section we first discuss the effect the precision has on the outcome of the algorithms. The precision relates to how many bits are kept in the conversion of the distinguishing scores to integer weights. After we have answered this question, we then run a series of experiments for both the key rank algorithm and key enumeration algorithm, comparing them to other schemes within the literature.

To run our tests we run emulations from a toolbox kindly provided to us by Veyrat-Charvillion. The toolbox emulates a leakage trace (only at the required points) using Hamming weight leakage with random noise added (i.e. sampled from a Gaussian distribution with zero mean and adjustable variance). For the DPA attack a template attack is performed by utilising template matching as a distinguisher (this has been shown by Mangard et al. [111] to be equivalent to performing a correlation based DPA attacks with a perfect model). The attack targets the output of the SubBytes operation in the first round.
6.5.1. Precision

In DPA there are a variety of factors which influence the effective precision of the distinguishing scores. These factors include: the number of traces, the way in which the power traces were measured, and the model; all of which will be briefly discussed here. The level of precision in the DPA scores will directly influence the precision required when the distinguishing scores are mapped to integers (discussed below). We then experimentally determine the required precision for our key rank algorithm and compare this to the number of bins required in the key rank algorithm of Glowacz et al. [78]. The number of bins in the algorithm by Glowacz et al. affects the accuracy of their algorithm and thus can be seen as an analogue to the number of bits in our key weights.

MapToWeight takes in probabilities and outputs additive integers to a fixed precision $\alpha$ such that the smaller the value is, the more likely it is. The first step is to take the (absolute value of the) logarithm which serves two purposes. Firstly, it converts the scores from multiplicative to additive and, secondly, makes the smaller scores more likely (since the values were all initially between 0 and 1). After this, if the largest value is approximately $2^\beta$, and if $\alpha$ bits of precision are required, multiplying all values by $2^{\alpha-\beta}$ and then rounding to integers will have the desired result and the scaling will not affect the relative likelihood of the scores. These values are then output as the integer weights. Formally $w_{i,j} = \text{MapToWeight}(d_{i,j})$ where $\text{MapToWeight}(d_{i,j}) = \lfloor \text{abs}(d_{i,j} \cdot 2^{\alpha-\beta}) \rfloor$ for $\alpha$ bits of precision.

6.5.2. Factors Influencing the Precision Levels in DPA

In this section we touch on, what we feel are, the three major factors which contribute to the amount of precision required in key rank to successfully capture all of (or at least a sufficient amount of) the information within a trace.

Power Model

Many practical DPA attacks use the Hamming weight or Hamming distance to map values to predicted power consumption, to form the power model. If Hamming weight is used on an eight bit implementation of AES-128 (as in our case), the power model can take values 0 through 2^8.
8. Hence, to capture the power model (in a perfect world where there is no noise), only four bits of precision would be required. Other models that are commonly studied include exploiting leakage of a single bit (normally either the most or least significant bit) which would only require a single bit of precision. The leakage model, where it is only possible to distinguish the zero byte from a non-zero byte, also only requires a single bit of precision. It is possible to construct leakage models which require more bits of precision to capture all possible values but these rarely occur in practice.

**Leakage Traces**

Modern oscilloscopes offer a vertical resolution of between 8 and 12 bits. While dedicated AC/DC converters may achieve up to 16 bits of vertical precision, they can only do so at low sampling frequencies and, hence, the standard choice is to use a lower resolution in favour of a higher sampling frequency. To capture the information output by the oscilloscope the key rank or enumeration algorithm requires precision at least the same as the oscilloscope. If several traces are averaged before the information is passed to the key rank algorithm then extra precision will be required. For example, if originally the oscilloscope can just output a single bit, then only 1 bit of precision is required but if two traces are averaged, the values taken can be 0, \(\frac{1}{2}\), 1 and therefore the key rank algorithm will need two bits of precision to capture the information.

**Distinguisher**

The distinguisher itself will have an impact on the precision required, based on the precision it outputs the distinguishing vector with. However, the precision of a distinguisher is hard to predict without empirical evidence, since it also depends on the precision of the data being input as well as the operations it performs. However, the distinguisher must still be taken into consideration when establishing how much precision is required by the key rank algorithm.

**6.5.3. Experimentally Measuring the Precision**

We ran precision tests using Veyrat-Charvillon’s emulator, described above, with 30 traces and variance of two to determine a suitable amount of precision to keep for the remaining experi-
Figure 6.8.: Bits of precision for Key Rank (left) and number of bins for Glowacz et al. [78] (right) [113]

ments. Figure 6.8 (left) shows the difference in ranking outcomes for increasing precision. In this figure (and all that follow), the individual experiments are plotted in grey while the average is plotted in black. The $x$ axis plots the precision of $w_{i,j}$ while the $y$ axis plots the change in the ranking outcome relative to the previous result. The precision experiments were run in increments of $0.1$ bits. It can be seen that the results converge at 11 bits of precision. Since our algorithm is exact, it can be concluded that with 11 bits of precision the key can be exactly ranked. With as few as five bits, the results are within 5 bits of accuracy and at eight bits the results are within 1 bit of accuracy. These results are already comparable with the other rank estimate algorithms.

For comparison, we implemented the convolution based rank estimation algorithm by Glowacz et al. [78]. The algorithm works by constructing $m$ histograms (one for each of the $m$ key chunks, using the distinguishing vector) and counting the keys by counting the number of items, in the final histogram, before the bin with the secret key in. The final histogram is constructed by convoluting all $m$ histograms together, one at a time. Figure 6.8 (right) shows that they achieve high precision for 50,000 bins (per initial histogram) onwards. Comparing this to our scheme, we conclude that 50,000 bins is roughly equivalent to using 11 bits of precision within the $w_{i,j}$.

Glowacz et al. [78] actually recommend 500,000 bins in total for AES-128, or 31,250 bins per histogram. From Fig. 6.8 we can see that this may in fact be a slight under-estimate.

We previously touched on the fact that we believe that different distinguishers would require
Figure 6.9.: Impact of the distinguisher (left: correlation, right: Gaussian templates without $\log_2$) on the precision requirements [113]

different precision levels to provide accuracy. To empirically verify this statement, we implemented two further distinguishers; the first was the same templating technique but without the use of logarithms, the second was performing a correlation based attack. Figure 6.9 shows the results of these experiments, allowing up to 16 bits of precision. From these graphs it can be seen that the correlation based attacks require the least amount of precision to converge on an accurate solution, while taking logarithms in the template attack increases the precision required. This experiment supports our previous claim that the attack itself must be considered when choosing the appropriate level of precision. For the remainder of the experiments, we continue to use the templating method including the logarithm and, therefore, this can be seen as the “worst case”, in that other methods require lower precision (or give a tighter estimate for the same precision).

To provide further evidence for the exactness of our ranking algorithm when given enough precision, instead of just considering the convergence of our algorithm, we considered the difference in estimated key rank between our algorithm and the algorithm by Glowacz et al. Since the error rate for their algorithm is known, it can be used as a baseline to consider the exactness of our algorithm. In this experiment we used 16 bits of precision for our algorithm and 500,000 bins per histogram for the Glowacz et al.’s algorithm. The result can be seen in Fig 6.10. The result shows an identical trend to that of Fig. 6.8 (right) implying that the difference between the two algorithms is that of rank estimation error given by Glowacz et al., acting as further evidence as to the exactness of our algorithm.
As explained in Sect. 6.3, the depth of the key does not appear in the time complexity of the key rank algorithm and therefore will not affect the runtime. The runtime of the algorithm depends on the size of the graph which is defined by \( n, m, \) and \( W \). For a given implementation of AES-128, \( n \) and \( m \) are fixed and thus the graph size is solely dependent on \( W \). Since \( W \) is defined by summing the weights of the secret key, the size of \( W \) will be affected by the precision kept within the weights \( w_{i,j} \) when they are converted from distinguishing scores.

Figure 6.11 shows the runtime of the key rank algorithm against the value of \( W \), while Fig. 6.12 shows the runtime against the amount of precision kept in the \( w_{i,j} \). To calculate these
graphs we fixed all parameters for Veyrat-Charvillon’s emulator and only varied the precision in the MapToWeight function. Both graphs could be generated by just varying the precision, since more precision gave larger values, leading to a larger \( W \). As with previous experiments, we upper bounded the precision of the \( w_{i,j} \) at 16 bits. As can be expected, the runtime is linear in \( W \) (this agrees with the runtime analysis) and is exponential in the \( w_{i,j} \) precision (an extra bit allows integer values to be twice as large). The runtimes for sufficient precision (8 bits) are below half a second while for 11 bits of precision (as shown to provide accurate results from the previous experiments) has an average runtime of 4 seconds. The graph gives the average value in black while the actual values recorded by the experiment are given in grey.

### 6.5.5. Key Enumeration

The runtime of the key enumeration algorithm is dominated by the depth of the key, unlike the key rank algorithm. This is because it has to store the partial keys, as it creates the keys to enumerate. In all graphs and results within this section, key enumeration does not perform an AES-128 encryption and perform a comparison with a given plaintext/ciphertext pair, but uses a known secret key and, therefore, can do a direct comparison. This technique is common in the literature and we use it for all methods tested to make for a fair comparison. The AES-128 encryptions will increase the timings but will not change the relative results; to enumerate \( B \) keys, all schemes will have to perform \( B \) AES-128 comparisons and, thus, adds a constant
Figure 6.13.: Comparison of the runtime of our algorithm and Veyrat-Charvillon et al.’s algorithm (single core) [113]

Figure 6.14.: Comparison of the runtime on a varying number of cores [113]

factor to all algorithms. Therefore, the results without the encryptions still hold.

Figure 6.13 gives a comparison of the runtimes of Veyrat-Charvillon et al.’s algorithm [163] and our key enumeration algorithm. We sampled multiple distinguishing scores at each key depth and ran both key enumeration algorithms. The graph shows that, from key depths of around 30 bits (rank $2^{30}$ onwards), our algorithm clearly outperform Veyrat-Charvillon et al.’s algorithm, even on a single core. Since our algorithm outperforms on a single core and is more parallelisable, from this point forward we focus on comparing our algorithm across multiple cores.

Figure 6.14 shows the result of running the key enumeration across multiple cores. From the graph it can be seen that, with eight cores $2^{40}$ keys can be enumerated, in the same time that
one core can enumerate $2^{37}$ keys. The result shows a reasonably linear improvement (which is optimal): approximately 8 times as many keys were enumerated when 8 cores were used instead of one. Figure 6.14 also shows that a single core enumerates $2^{38}$ keys in 13.9 hours while four cores perform the same enumeration in 6.4 hours; approximately twice as fast. While the algorithm is linear, per core, in the number of keys enumerated in a fixed period of time, it is not linear, per core, in the time taken to enumerate a fixed number of keys. This is because the width $W$ of the graph does not decrease (for the core enumerating the deepest key) when the number of cores is increased; only the number of keys enumerated per core is decreased. Therefore, the weight $W$ is adding a cap to the amount parallelism can help.

6.6. Relating Key Rank and Leakage Resilience

In this section we take the key rank algorithm designed within this chapter and use it to try and garner insight into how real devices behave with respect to the number of bits leaked per call. Unfortunately, there is disconnect between the leakage resilient work within this thesis and the capabilities offered by our key rank algorithm. The key rank algorithm currently only works when the key chunks are independent. However, all the leakage resilient research carried out within this thesis are ideally suited to being instantiated over elliptic curves. Elliptic curves do not have a (known) way to be expressed such that there is a notion of independent key chunks. Thus, we must consider a different scheme within the leakage resilient model, such that the underlying components do have independent key chunks. We have chosen to investigate the leakage resilient PRG by Pietrzak [133] for this purpose, as it can be instantiated using AES. Below we recap the scheme and discuss the leakage bounds given in the paper, before using the leakage emulator, used previously in this chapter, to practically investigate the amount of leakage and conclude by discussing our findings.

All the simulatable leakage work discussed in Ch. 4 is built upon AES. Therefore, any insights gained as to how AES leaks and at what rate, may help the design of a practical simulator, which is not currently known to exist. On the other side, better understanding of the behaviour of the AES leakage will make simulators harder to construct, since it gives another known aspect for a
6.6.1. Theoretical Bounding of Leakage

The scheme that we will consider is the leakage resilient PRG by Pietrzak [133]. The scheme is built using a wPRF as a building block and we have chosen to instantiate it with AES-128, making it a suitable candidate for use with the key rank algorithm.

A diagram for the scheme is given in Fig. 6.15. The scheme has two keys $k_0, k_1$ and the keys are used in an alternating fashion. Due to the OCLI assumption, future computation attacks are prevented (See Sect. 3.6.2 for more details on the OCLI assumption). Not only does each call generate some pseudorandomness, it also updates the key. This means that each key can only be leaked on twice; once when it is generated and once when it is used. The theorem statement is then given below.

**Theorem 16** ( [133, Theorem 2]). Let $A$ be an adversary against the leakage resilient PRG security of $G$, then there exists an adversary $B$ (of similar complexity to $A$) against the wPRF security of $F : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^{n+n}$ such that:

$$
\text{Adv}_G^{\text{lr-prg}}(A) \leq 8 \cdot q \cdot (\text{Adv}_F^{\text{wprf}}(B))^{\frac{1}{12}}
$$
where $q$ is the number of queries to $G$, the leakage is defined as $\lambda = \frac{\log(\text{Adv}_{wprf}(B)^{-1})}{6}$ and if $\text{size}(B)$ denotes the size of the adversary $B$ then $\text{size}(A) = \frac{(\text{Adv}_{wprf}(B)^{2}\cdot\text{size}(B))}{2^{\lambda^2}(n+\kappa)^4}$.

The $\text{lr}$-PRG notion used here is similar to the one discussed in Sect. 4.1 except the adversary can choose the leakage function adaptively per call instead of it being fixed at the start of the experiment. The size function represents the adversary’s size in terms of being represented as a circuit. Since we do not consider an adversary’s size as a circuit throughout this work, it will not be discussed in detail here.

From the given leakage bound it can be seen that the wPRF advantage can be rewritten as $\text{Adv}_{wprf}(B) = 2^{-6\cdot\lambda}$. Substituting this back into the PRG advantage results in $\text{Adv}_{LR-prg}(A) \leq 8 \cdot q \cdot 2^{-\lambda}$. However, this advantage states that the more leakage the adversary is allowed, the more secure the PRG is. This clearly does not relate to how a device behaves in practice, where, the more a device leaks, the easier it is to attack and, thus, we consider another equality given in the theorem statement to get a more suitable bound. Rearranging the size equality and treating the adversary sizes as constants gives $\text{Adv}_{wprf}(B) = c \cdot 2^{\lambda+2}$ for a suitable constant $c$. Substituting this into the advantage of $A$ gives $\text{Adv}_{LR-prg}(A) \leq c' \cdot q \cdot 2^{\lambda}$ which can be rewritten again to give $\lambda \leq 24 \cdot (\log(\text{Adv}_{LR-prg}(A)) - \log(c \cdot q))$.

It is interesting to note that this form looks extremely similar to the leakage description given in the leakage resilient literature when elliptic curves are used [75, 101]. These similarities give us hope that anything discovered here is also relevant to leakage resilience on elliptic curves. One question that arises from bounds such as this is with regard to the number of queries. On the $i$th leakage query does the amount of leakage correspond to the current query or the maximal query? Both of these interpretations have issues. If the query term corresponds to the maximal number of queries that a scheme is secure against, $j$ queries does not imply a scheme that is secure against $j-1$ queries (since the scheme with $j-1$ queries is allowed more leakage than the scheme with $j$ queries). This is extremely counter-intuitive. However, if the other interpretation is considered, where the variable corresponds to the current query number, implies that each query gives a different amount of leakage which does not appear to model how a device behaves in practice. While there are still issues with the continuous leakage model, it is one of the models which most closely models how a device behaves in practice. Since one interpretation
of the leakage is not clearly more desirable than the other we will investigate both.

In Fig. 6.16 we plot the number of queries versus the total amount of leakage for both of these options. Parameters were chosen so that the leakage scales in the same manner and to the same value as an attack would. Since we are only interested in the asymptotics of the leaks and not the concrete parameters we will not discuss them here. Since we are mainly focusing on an adversary recovering the key (the adversary’s CKR-UL advantage), as this is what can be compared against in a practical setting, the amount of leakage extracted by the adversary will be bounded by the length of the key. To model leakage on the key, when the calculations state that the key has been completely leaked (when 128 bits have been leaked, since we are using AES-128), from this point forward the leakage will be 128 bits, regardless of if the formula would allow more.

6.6.2. Practical Bounding of Leakage

We now perform practical experiments to contrast against the theoretical results above. For a given secret key, we will generate a plaintext and encrypt it, generating leakage. The leakage will then be used to generate a distinguishing vector which can be used to rank a key. This process can then be repeated for a new plaintext, giving more leakage to update the distinguishing vector. The attack used will be the same template attack used within the previous section for testing the key rank and key enumeration algorithms. This whole process will be repeated multiple times.
In practical scenarios, the key rank algorithm can be used as a measure of leakage when the algorithm is adjusted to rank the secret key last amongst equally weighted keys. If there are $K$ keys total and key rank returns a rank of $R$ then we can set the leakage to be $\lambda = \log K - \log R$. In the case where the key distinguishing vector is almost uniform, results in $R$ and $K$ being extremely similar and thus the amount of leakage is close to zero, while when the key is ranked top, gives $\lambda = \log K$, showing that the key has completely leaked.

Using the key rank to calculate our leakage metric, in the experiment described above, we used the leakage emulator from the implementation section to collect results for different noise levels (varying the signal to noise ratio). In Fig. 6.17 the results of this experiment can be seen.

Every time a new trace was collected, the DPA attack was rerun and the key rank was recalculated. 500 repeat experiments were performed for each signal-to-noise ratio, with the number of traces used in the correlation increasing from 1 up to 100. It can be seen that all of the graphs follow a similar pattern, increasing in a logarithmic manner until no more information can be leaked as the key is completely known. It can be seen that the noise slows the increase of the key rank and therefore lowers the amount of (useful) leakage received each round. This corresponds directly with what happens in practice, where lower signal-to-noise makes a device harder to attack [110]. While the noise increasing lowers the amount of leakage, the graph stays logarithmic in shape.

It is important to note that the above results are only for power traces utilising a templating attack. Other attacks or side channels are not guaranteed to give similar results. The experiment

![Figure 6.17: $\lambda$ leakage compared to the number of traces](image)
would have to be run for each attack and side channel separately. The goal is to be able to run the experiment once, for a particular side channel and attack, and argue why these results extend to a large class of attack methodologies. We chose to investigate power as a side channel with a templating attack as these are both extremely common within the literature.

### 6.6.3. Comparing Both Leakage Bounds

As can be seen from Figs. 6.16 and 6.17, there are similarities between the theoretically and practically measured leakage. Both graphs increase in a logarithmic fashion until the key has been completely leaked. Figure 6.18 compares them both directly. The figure shows that the leakage bounds given in the theoretical paper are not just a relic of the proof techniques but, in fact, relate to how leakage behaves in practice. Since the leakage resilient papers involving elliptic curves [75,101] also have a similar form of leakage, it gives credence to the fact that this may also be an accurate representation of how leakage occurs on an elliptic curve. To confirm this fact, a similar test would have to be devised that can work on elliptic curves where there is no notion of independent key chunks.

The experiments in this section demonstrate that leakage does behave as given within the theorem statements. These experiments can now be used to test if a device meets the leakage bounds given within a theorem. It is now possible, using the technique discussed above, for a
hardware designer to test if their implementation of AES meets the \( \lambda \) leakage requirements set out by the scheme.

While this is a step in the right direction, the test above is only for a single style of attack. Using different style attacks will reveal a different amount of leakage based on the success of the attack. Thus, a designer can only state that a device’s leakage is below the given bound for all attacks that have been tested. It may be possible that a new attack is discovered which breaks the bound by leaking too much. However, this is still a step in the right direction and the first experiment which allows a designer to verify their bounds.

Another question is that, while a designer can run the experiments above for their specific scheme on a given piece of hardware and get a similar graph to those above, how do they convert this into a leakage bound \( \lambda \)? One option is to take the average leakage per trace. If there is one trace which contains lots of leakage, this could be enough to extract the key, even if the other traces contain no leakage information. Therefore, a more sensible metric may be the maximum leakage across all traces. Using this metric if a scheme is secure against the maximum amount of leakage per trace, it is guaranteed to be secure. However, while the change in the key rank can be seen as a metric for the number of bits of leakage, it is not clear how this can necessarily instantly translate to exploitable leakage, especially when a key update mechanism is being used. While this may not be the perfect solution, it is one of the first works to be able to capture the leakage bound.

6.7. Conclusion

In this chapter we constructed algorithms for both key rank and key enumeration when the key chunks are independent. In this scenario our algorithms are the fastest rank key to arbitrary depths, even when providing a higher amount of accuracy than other algorithms. For key enumeration, our algorithm is the fastest to enumerate to a given depth and can enumerate more keys than other algorithms in a fixed time period. When the precision is set to be suitably high (approximately 11 bits) we provide the first key rank algorithm which can rank the key exactly, instead of just estimating the key rank, as previous work did.
Most block ciphers have the property that each key chunk is independent, and the leakage on the key chunks is independent. This is not always the case, for example many public key cryptography schemes do not meet the independence requirement. Future work for key enumeration would be to try and create key rank and enumeration algorithms which support non-independent key chunks. Lange et al. [107] present a method to enumerate likely key candidates after obtaining side channel information from a Diffie-Hellman key exchange over elliptic curves. They briefly discuss how to extend their method to give a rank estimation algorithm. However, ideally an exact ranking algorithm is required for calculating a leakage bound, so that a designer can be sure their implementation meets the given leakage bound. While this paper is the first to look at ranking for public key cryptosystems, it will need to be extended before it can be used for our purpose. Being able to rank (and enumerate) keys for (arbitrary) non-independent key chunks would open up a whole host of possible applications. One such application is in the recent attacks on RC4 [77,128,129]. In some of these attacks the goal is to recover a user’s password for a particular website using a priori information on password distributions from previous password attacks. In this scenario passwords tend to be words, or easily memorable phrases, and hence each letter of the password is not independent from the last. Therefore, a key enumeration algorithm would have to be able to cope with this a priori information. Secondly, the biases within RC4 (can be considered as the leakage) are pairwise dependent on the bytes and therefore the leakage is not independent per byte. If a key rank algorithm was constructed that deals with non-independent key bytes it could be applied to the RC4 scenario. A key enumeration algorithm would be able to make the RC4 attacks more efficient than the standard testing methodologies currently used.

Using our exact key rank algorithm, we showed that the theoretical bounds given within leakage resilient papers which rely upon AES [133] closely match the amount of leakage which can be extracted from a series of traces. Unfortunately we cannot apply the same technique to elliptic curves due to the dependencies in the leakage. However, due to the similarity of the leakage bounds for the leakage resilient PRG, and the schemes which are built upon elliptic curves [75, 101], we conjecture that leakage on an elliptic curve will behave in a similar manner in practice.
The ability to rank keys with dependent key chunks would allow the leakage resilience research within this thesis to be studied in greater detail. Where we performed leakage tests on the leakage resilient PRG [133] and conjectured that elliptic curves should behave the same, the new key rank algorithm would allow leakage testing to be carried out on elliptic curves to see how they leak. Such an algorithm would allow the Kiltz Pietrzak key update mechanism [101] to be studied in more detail to see how it leaks and how different implementation techniques affect the amount of leakage. As discussed in Ch. 5, by understanding and reducing the leakage in the key update mechanism, a whole suite of leakage resilient schemes will have their leakage reduced.
7. Conclusion

After reviewing the current leakage resilient models that are available in the literature (Ch. 3), we focused on the two models which are the most desirable balance of theory and practice. The first model chosen was the simulatable leakage model (Ch. 4). The ability of an evaluation lab to implement the constructions in hardware and test the validity of the assumptions made, makes this model desirable. The other advantage is that a mathematical definition of the leakage set is not required. When a leakage set is formally defined they tend to be overly powerful, or not capture how a device behaves in practice. Since the simulator has direct access to a device, no such set will need to be given. The simulatable leakage model shows that a scheme is secure if a simulator (which does not have access to the secret information) can be constructed that can simulate the leakage. The scheme is secure if all adversaries cannot distinguish between real and simulated traces. In this work we break the only known simulator against AES-128 by constructing a distinguisher using cross-correlation. Since our work, there have been papers constructing schemes which are secure in the face of leakage providing the existence of a simulator and thus it becomes even more pressing to construct such a simulator. Without new techniques, it appears that no such simulator can exist, as backed up by our (failed) attempts at fixing the simulator. Without the existence of a simulator, any reductions to a simulator no longer provide any security guarantees. Given this we considered a second leakage model.

The second model considered was the continuous leakage model using the OCLI assumption (Ch. 5). While it has been shown that, in certain circumstances, the OCLI assumption does not hold, for the specific cases considered within this work, this model closely resembles how devices leak in practice. The model allows for a bounded amount of leakage per function call, while allowing an arbitrary amount of total leakage. This is in contrast to other models which
only allow a total amount of leakage (regardless of the number of calls to the device). Therefore, the continuous leakage model more closely resembles the majority of practical attacks (including DPA and EM attacks). The class of leakage functions is too strong, as it allows attacks that cannot occur in practice (such as the future computation attack). However, unlike other leakage sets (such as \( \mathcal{AC}_0 \)) it captures the majority of practical attacks. There are certain scenarios where the OCLI assumption does not hold. Yet, if the two parts of the computation are suitably separated in time, the assumption tends to hold in practice. In this model we began by defining several security notions for the key update mechanism against leakage. The computational variation (CKR-UL) is a necessary condition for security to achieved. If the adversary can learn the key from the leakage of the update function then no security will be possible for any scheme which uses the key update function. The decisional variation (DKR-UL) shows that an adversary can learn no information about the secret key. We argue why a key update mechanism which meets the DKR-UL security property would allow highly modular proofs. The leakage resilient security of a scheme could be reduced to the DKR-UL security of the key update mechanism plus the non-leakage security of the scheme. We leave it as an open question to construct such a key update mechanism. We then constructed a pair of leakage resilient MACs, a leakage resilient PRF and leakage resilient (authenticated) encryption. To construct these schemes we used new proof techniques, as well as proving some generic composition theorems for leakage resilience. The generic compositions enable any leakage resilient components to be assembled to create more complicated leakage resilient schemes. This is some of the first work to provide such desirable composition theorems. To prove the schemes secure with leakage (which can then be used in the composition theorems) we use new proof techniques. These proof techniques help to simplify
the proving of security against leakage in the GGM. One of the important open questions is, if a
MAC can be constructed which allows leakage on both tagging and verification? This could be
used directly within the generic composition to provide authenticated encryption which allows
more leakage. Since the writing of this work, we believe the VRFY∗ given in Fig. 7.1 would
allow verify to also leak in our MAC, due to the one way properties of the hash function H2.

We leave showing the security as future work. Another important question is what other leakage
resilient schemes can be constructed? Constructing a key agreement protocol would allow an
entire leakage resilient SSL ciphersuite to constructed. If the key agreement protocol was built
upon the same key update mechanism, the entire SSL suite would be built upon the same key
update mechanism, providing a single scheme to be studied further and to investigate the best
way to reduce its leakage. We are the first work to implement some of these leakage resilient
schemes and investigate their practical efficiency and leakage. We have discussed how randomness
generation affects the amount of bits leaked per call but there could be several other similar
considerations that are required to be made, to implement these schemes in practice.

One of the biggest open questions is; is there a better leakage resilient model? There are mul-
tiple models but they are either too powerful or too weak. Any model which does not completely
capture a real world adversary’s capabilities runs the risk of providing a proof of security but
still being broken when implemented. This is a similar scenario to proofs that were given before
side channels were discovered. The scheme was assumed secure but subsequently broken when
the extra information was utilised. Overly powerful models capture attacks that occur in practice
but may also capture attacks that cannot occur in practice (such as future computation attacks).
The resulting schemes are then unnecessarily complex to protect against these attacks, despite
them being relics of the model. By creating a model close to practice, it will capture all realistic
attacks while excluding attacks that cannot occur in practice. In this new model schemes can
then be created that are hopefully both secure and efficient.

There are currently two desirable models to theoretically capture side channels. The first is
the continuous leakage model utilising the OCLI assumption, as used throughout this thesis.
The second is the probing model [92], as discussed in Sect. 3.2. The former model is too strong
(capturing future computation attacks), while the latter is too weak (e.g. does not capture timing
attacks). There has been recent work making the probing model more powerful [58, 68, 142]. In this thesis we showed how to construct schemes using the OCLI assumption which are more practical. Of the two models, the probing model, along with its connections to the noisy leakage model, has become the favoured out of the two [17, 42, 63]. This is in part due to the efficiency of the schemes within the model at capturing attacks which utilise side channels such as power or EM and partly due to the difficulty of trying to estimate the leakage of a construction in the continuous leakage model (although we made the first steps towards answering this question in Ch. 6). The question arises, is there a model which sits between the two? This would allow more powerful attacks to be captured, without the inefficiency caused by the continuous leakage model. If such a model does not exist it may be possible to construct schemes that are secure in both the continuous leakage and probing models. The proof in the continuous leakage model would capture the most powerful adversaries. The probing model proofs should allow more leakage while removing requirements, such as the GGM. The proofs in the probing model would also clarify the effect changing the number of shares has on the security of a scheme, as this is currently not captured by proofs in the continuous leakage model. Giving both proofs enables implementers to make informed decisions about the schemes being chosen.

The thesis is concluded with an attempt to quantify the leakage of a device, to relate it to the leakage bound provided by the theoretical models (Sect. 6.6). To achieve this we provide (in Ch. 6) the first key rank algorithm which can exactly rank a known secret key (given enough precision). Previous algorithms could only estimate the rank of the key. Using traditional algorithmic book keeping techniques, we were able to convert the key rank algorithm to provide an efficient key enumeration algorithm. Experiments show that our algorithms are comparable in speed to the best algorithms in the literature [78, 163], despite the increased accuracy. For enumeration beyond $2^{26}$ our algorithm is more efficient. Using our ranking algorithm, it is possible to investigate the leakage of a device. The secret key is ranked repeatedly after more traces are added to the side channel attack. The key’s rank increasing, as more traces are added, can be used as an estimate of the amount of leakage on the underlying secret key. Comparing this to the theoretical leakage allowed on the leakage resilient PRG [133], we see that the theoretical bound is not just a proof relic but closely relates to how a device leaks in practice. This is the
first work to investigate the theoretical bounds given within the literature and compare how they related to leakage observed from a device. The key rank algorithm requires that each key chunk is independent. An open research question is how to construct an algorithm that does not have this restriction. The new algorithm would allow a similar investigation to be performed on the leakage resilient schemes constructed in this work.

A further research direction is how the key rank relates to entropy. If it can be shown that the key rank can be used as (a suitable approximation of) an entropy then this opens up a whole host of possible advantages. One use is as further investigation into block ciphers and their behaviour across repeated side channel attacks or as the amount of side channel information in the attack is increased. From a leakage resilient perspective, more information can be gained about the entropy of a secret key as the scheme runs. This will allow a lot more information to be gained about the leakage of scheme, instead of just the number of bits leaked from the key. For example, it will allow experiments to confirm the security when the proofs allow the entropy to be reduced, instead of a fixed number of bits leaking.

7.1. Publications

7.1.1. In This Thesis

Here we list all papers that relate to the work carried out in the thesis.

Simulatable Leakage: Analysis, Pitfalls, and New Constructions

Jake Longo Galea, Daniel P. Martin, Elisabeth Oswald, Daniel Page, Martijn Stam and Michael Tunstall


Abstract: In 2013, Standaert et al. proposed the notion of simulatable leakage to connect theoretical leakage resilience with the practice of side channel attacks. Their use of simu-
tors, based on physical devices, to support proofs of leakage resilience allows verification of underlying assumptions: the indistinguishability game, involving real vs. simulated leakage, can be ‘played’ by an evaluator. Using a concrete, block cipher based leakage resilient PRG and high-level simulator definition (based on concatenating two partial leakage traces), they included detailed reasoning why said simulator (for AES-128) resists state-of-the-art side channel attacks.

In this paper, we demonstrate a distinguisher against their simulator and thereby falsify their hypothesis. Our distinguishing technique, which is evaluated using concrete implementations of the Standaert et al. simulator on several platforms, is based on ‘tracking’ consistency (resp. identifying simulator inconsistencies) in leakage traces by means of cross-correlation. In attempt to rescue the approach, we propose several alternative simulator definitions based on splitting traces at points of low intrinsic cross-correlation. Unfortunately, these come with significant caveats, and we conclude that the most natural way of producing simulated leakage is by using the underlying construction ‘as is’ (but with a random key).

This work corresponds to the simulatable leakage research in Ch. 4.

**Counting Keys in Parallel After a Side Channel Attack**

*Daniel P. Martin, Jonathan F. O’Connell, Elisabeth Oswald and Martijn Stam*


**Abstract:** Side channels provide additional information to skilled adversaries that reduce the effort to determine an unknown key. If sufficient side channel information is available, identification of the secret key can even become trivial. However, if not enough side information is available, some effort is still required to find the key in the key space (which now has reduced entropy). To understand the security implications of side channel attacks it is then crucial to evaluate this remaining effort in a meaningful manner. Quantifying this effort can be done by
looking at two key questions: first, how ‘deep’ (at most) is the unknown key in the remaining key space, and second, how ‘expensive’ is it to enumerate keys up to a certain depth?

We provide results for these two challenges. Firstly, we show how to construct an extremely efficient algorithm that accurately computes the rank of a (known) key in the list of all keys, when ordered according to some side channel attack scores. Secondly, we show how our approach can be tweaked such that it can be also utilised to enumerate the most likely keys in a parallel fashion. We are hence the first to demonstrate that a smart and parallel key enumeration algorithm exists.

This paper designs the key rank and enumeration algorithms given in Ch. 6 and compares it to other schemes.

A Leakage Resilient MAC

Daniel P. Martin, Elisabeth Oswald, Martijn Stam and Marcin Wójcik


Abstract: We put forward the first practical Message Authentication Code (MAC) which is provably secure against continuous leakage under the Only Computation Leaks Information (OCLI) assumption. Within the context of continuous leakage, we introduce a novel modular proof technique: while most previous schemes are proven secure directly in the face of leakage, we reduce the (leakage) security of our scheme to its non-leakage security. This modularity, while known in other contexts, has two advantages: it makes it clearer which parts of the proof rely on which assumptions (i.e. whether a given assumption is needed for the leakage or the non-leakage security) and it also means that, if the security of the non-leakage version is improved, the security in the face of leakage is improved ‘for free’. We conclude the paper by discussing implementations; one on a popular core for embedded systems (the ARM Cortex-M4) and one on a high end processor (Intel i7), and investigate some performance and security aspects.

This paper formalises the key update mechanism of Kiltz and Pietrzak [101] and constructs a leakage resilient MAC in the same model. Our scheme is then compared to other schemes in
terms of the number of operations performed and the leakage is practically investigated. This can be found in Ch. 5.

**LeakAE**

*Daniel P. Martin, Elisabeth Oswald, and Martijn Stam*

In: Draft

**Abstract:** We put forward the first leakage resilient Authenticated Encryption scheme in the continuous leakage model using the only computation leaks information assumption. This scheme has several advantages over other schemes available in the literature. Firstly it is in the continual leakage model; while the majority of schemes in the literature only allow a total bounded amount of leakage, we allow a bounded amount of leakage per call but an arbitrary amount of total leakage. Secondly we are one of the few schemes that allows encryption of arbitrary length messages, not only that but we achieve this with a fixed (and fairly small) stretch constant. Finally we are the first scheme to provide nonce based authenticated encryption in the face of leakage.

Along the way we prove several interesting composition theorems still hold in the face of leakage. We show that a leakage resilient PRF in CFB mode gives a leakage resilient encryption scheme. From the authenticated encryption side, we show that the majority of the general compositions (using PRF and encryption) are not secure when leakage is involved. However two of the general compositions are still secure and allow leakage resilient authenticated encryption to be created.

We construct a leakage resilient PRF with the desired properties to allow the general compositions listed above to be instantiated. While in the non-leakage world only a PRF and encryption is needed, we show that in the leakage world a leakage resilient MAC is also needed because a leakage resilient PRF does not instantly give a leakage resilient MAC.

This paper constructs the leakage resilient authenticated encryption scheme, using generic composition of leakage resilient components, given in Ch. 5.
7.1.2. Other Publications

Here any other work done during the course of my Ph.D. which does not directly relate to the content of this thesis is listed.

**Characterisation and Estimation of the Key Rank Distribution in the Context of Side Channel Evaluations**

*Daniel P. Martin, Luke Mather, Elisabeth Oswald, and Martijn Stam*

In: To appear at ASIACRYPT 2016

**Abstract:** Quantifying the side channel security of a device has been a significant question within the community for several years. We show, using a highly accurate key rank algorithm, and based on using leakage traces of a complex real-world target, that the distribution of the key rank has a large variance and skew precisely in the range where enumeration can make a real difference. This implies that there is a huge, practically significant, gap between lucky outcomes (best case adversaries) and worst case outcomes, and consequently metrics attempting to capture any notion of attack success have to be thoughtfully chosen. Using carefully crafted simulations, we provide conclusive evidence that this observed rank behaviour is not specific to our example: we can observe the same behaviour across different distinguishers and signal-to-noise ratios. We conclude our research by connecting non-parametric order statistics, in particular percentiles, in a practically meaningful way with business goals.

**How low can you go? Using side-channel data to enhance brute-force key recovery**

*Daniel P. Martin, Luke Mather, Elisabeth Oswald, Martijn Stam, Jake Longo and Benjamin Sach*

In: Draft

**Abstract:** Side-channel analysis techniques can be used to construct key recovery attacks by observing a side-channel medium such as the power consumption or electromagnetic radiation of a device while it performing cryptographic operations. These attack results can be used as
auxiliary information in an enhanced brute-force key recovery attack, enabling the adversary to enumerate the most likely keys first.

We use algorithmic and implementation techniques to implement a time- and memory-efficient key enumeration algorithm, and in tandem identify how to optimise throughput when bulk-verifying quantities of candidate AES-128 keys. We then explore how to best distribute the workload so that it can be deployed across a significant number of CPU cores and executed in parallel, giving an adversary the capability to enumerate a very large number of candidate keys.

We introduce the tool LABYNKYR, developed in C++11, that can be deployed across any number of CPUs and workstations to enumerate keys in parallel. We conclude by demonstrating the effectiveness of our tool by successfully enumerating $2^{48}$ AES-128 keys in approximately 30 hours using a modest number of CPU cores, at an expected cost of only 700 USD using a popular cloud provider.
Bibliography


262


[18] Sonia Belaïd, Vincent Grosso, and François-Xavier Standaert. Masking and leakage-resilient primitives: One, the other(s) or both? *Cryptography and Communications*, 7(1):163–184, 2015.


[27] Daniel J. Bernstein, Tanja Lange, and Christine van Vredendaal. Tighter, faster, simpler


[34] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the Weil pairing.


[42] Claude Carlet, Emmanuel Prouff, Matthieu Rivain, and Thomas Roche. Algebraic decomposition for probing security. In Rosario Gennaro and Matthew J. B. Robshaw, editors,


[64] Stefan Dziembowski and Ueli M. Maurer. On generating the initial key in the bounded-storage model. In Christian Cachin and Jan Camenisch, editors, Advances in Cryptology -


[71] Benjamin Fuller and Ariel Hamlin. Unifying leakage classes: Simulatable leakage and pseudoentropy. In Anja Lehmann and Stefan Wolf, editors, ICITS 15, volume 9063 of


[79] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or
A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th
Press.

[80] Shafi Goldwasser and Yael Tauman Kalai. On the impossibility of obfuscation with aux-
IEEE Computer Society Press.

[81] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *Journal of Computer and

[82] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of

[83] Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leak-

[84] Gilbert Goodwill, Benjamin Jun, Benjamin Jaffe, and Pankaj Rohatgi. A testing method-
ology for side channel resistance validation. NIST non-invasive attack testing workshop,
2008.

[85] Parikshit Gopalan, Adam Klivans, Raghu Meka, Daniel Stefankovic, Santosh Vempala,
and Eric Vigoda. An FPTAS for #knapsack and related counting problems. In Rafail
Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science,
FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 817–826. IEEE Com-
puter Society, 2011.

[86] Shay Gueron. Intel’s SGX memory encryption engine. Talk, Real World Cryptography
2015, Standford, USA, 2015.

[87] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul,


275


276


282


A. AES

AES is based on the Rijndael block cipher after its standardisation in the Advanced Encryption Standard [125]. The cipher was standardised in 2001, by NIST to replace the Data Encryption Standard (DES). The cipher was standardised with three key sizes 128, 196 and 256 bits. Each uses a plaintext of size 128 bits. In this thesis we use AES-128 and thus will give the algorithm for AES-128 below.

A.1. Algorithm

The key and the plaintext are 128 bits. The strings can be represented as a $4 \times 4$ byte array:

\[
\begin{pmatrix}
S_0 & S_4 & S_8 & S_{12} \\
S_1 & S_5 & S_9 & S_{13} \\
S_2 & S_6 & S_{10} & S_{14} \\
S_3 & S_7 & S_{11} & S_{15}
\end{pmatrix}
\]

The plaintext, in this form shall be denoted the state matrix. The AES-128 encryption algorithm is then given in Fig. A.1. After completion the state matrix is converted back into a bitstring to be returned as the ciphertext. Decryption can be defined similarly but will not be given here.

A.1.1. KeyExpansion

The KeyExpansion algorithm generates round keys from the master secret key. As the KeyExpansion algorithm is not considered in this work, we do not discuss it here.
A.1.2. AddRoundKey

The AddRoundKey step is the only step which adds secret information into the state. The algorithm works by exclusive-oring each byte of the round key with each byte of the state.

A.1.3. SubBytes

The SubBytes step provides the non-linearity in the algorithm. Non-linearity helps to prevent certain classes of attacks. The SubBytes step works by applying the SBox function to each byte in the state. The SBox was chosen to avoid fixed points and opposite fixed points. The SBox is normally given as a look-up table (see Table A.1) but can be defined algebraically. Let the function $g(x)$ be the inverse function in $GF(2^8)$ and the function $f(x)$ be defined as the following matrix operations:
Table A.1.: The AES SBox lookup table. The column is determined by the least significant nibble and the row by the most significant nibble

\[
\begin{array}{cccccccccccccccc}
00 & 01 & 02 & 03 & 04 & 05 & 06 & 07 & 08 & 09 & 0a & 0b & 0c & 0d & 0e & 0f \\
00 & 63 & 7c & 77 & 7b & f2 & 6b & 6f & c5 & 30 & 01 & 67 & 2b & fe & d7 & ab & 76 \\
10 & ca & 82 & c9 & 7d & fa & 59 & 47 & f0 & ad & d4 & a2 & af & 9c & a4 & 72 & c0 \\
20 & b7 & fd & 93 & 26 & 36 & 3f & f7 & cc & 34 & a5 & e5 & f1 & 71 & d8 & 31 & 15 \\
30 & 04 & c7 & 23 & c3 & 18 & 96 & 05 & 9a & 07 & 12 & 80 & e2 & eb & 27 & b2 & 75 \\
40 & 09 & 83 & 2c & 1a & 1b & 6e & 5a & a0 & 52 & 3b & d6 & b3 & 29 & e3 & 2f & 84 \\
50 & 53 & d1 & 00 & ed & 20 & fe & b1 & 5b & 6a & cb & be & 39 & 4a & 4c & 58 & cf \\
60 & d0 & ef & aa & fb & 43 & 4d & 33 & 85 & 45 & f9 & 02 & 7f & 50 & 3c & 9f & a8 \\
70 & 51 & a3 & 40 & 8f & 92 & 9d & 38 & f5 & bc & b6 & da & 21 & 10 & ff & f3 & d2 \\
80 & cd & 0c & 13 & ec & 5f & 97 & 44 & 17 & c4 & a7 & 7e & 3d & 64 & 5d & 19 & 73 \\
90 & 60 & 81 & 4f & dc & 22 & 2a & 90 & 88 & 46 & ee & b8 & 14 & de & 5e & 0b & db \\
a0 & e0 & 32 & 3a & 0a & 49 & 06 & 24 & 5c & c2 & d3 & ac & 62 & 91 & 95 & e4 & 79 \\
b0 & e7 & c8 & 37 & 6d & 8d & d5 & 4e & a9 & 6c & 56 & f4 & ea & 65 & 7a & ae & 08 \\
c0 & ba & 78 & 25 & 2e & lc & a6 & b4 & c6 & e8 & dd & 74 & 1f & 4b & bd & 8b & 8a \\
d0 & 70 & 3e & b5 & 66 & 48 & 03 & f6 & 0e & 61 & 35 & 57 & b9 & 86 & c1 & 1d & 9e \\
e0 & e1 & f8 & 98 & 11 & 69 & d9 & 8e & 94 & 9b & 1e & 87 & e9 & ce & 55 & 28 & df \\
f0 & 8c & a1 & 89 & 0d & bf & e6 & 42 & 68 & 41 & 99 & 2d & 0f & b0 & 54 & bb & 16 \\
\end{array}
\]

\[f(x) = \begin{pmatrix}
1 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 1 & 1 \\
1 & 1 & 1 & 0 & 0 & 0 & 1 & 1 \\
1 & 1 & 1 & 1 & 0 & 0 & 0 & 1 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 1 & 1 & 1 & 1 & 1
\end{pmatrix}
\begin{pmatrix}
x_0 \\
x_1 \\
x_2 \\
x_3 \\
x_4 \\
x_5 \\
x_6 \\
x_7
\end{pmatrix}
\begin{pmatrix}
1 \\
1 \\
0 \\
0 \\
0 \\
1 \\
1 \\
0
\end{pmatrix}
\]

where \(x = (x_0 \ldots x_7)\) is the bit representation of \(x\). The SBox of \(x\) is the defined as \(SBox(x) = f(g(x))\).

**A.1.4. ShiftRows**

In the ShiftRows algorithm the \(i^{th}\) row of the state matrix is (cyclically) left shifted by \(i\) columns.

This gives the following transformation:
The ShiftRows step helps to remove the linear independence of columns. Without this step, AES would be equivalent to applying four independent block ciphers, one per column.

**A.1.5. MixColumns**

The MixColumn algorithm adds diffusion to the algorithm. Diffusion is required so that each bit of the output is dependent on the majority of the input bits. Diffusion is achieved by multiplying each column by the following matrix (multiplication takes place in the finite field $GF(2^8)$).

\[ \begin{pmatrix} 2 & 3 & 1 & 1 \\ 1 & 2 & 3 & 1 \\ 1 & 1 & 2 & 3 \\ 3 & 1 & 1 & 2 \end{pmatrix} \]
B. Generic Group Model

B.1. Generic Group Model Oracles

In this Appendix we provide the generic group oracles required within the proofs. The oracles \( O_2, O_3, \text{Sample}_2, \text{Sample}_3, \text{Guess}_2, \text{Guess}_3 \) have been excluded due to their similarities to the oracles given in Fig. B.1. The sample routine performs \( \xi \leftarrow \Xi_1 \setminus L_1 \) which states that a random encoding string is chosen with the restriction that it is the first time an encoding is used. This prevents multiple distinct elements having the same encoding. However, it is possible that a single element has multiple encodings after the polynomials have been evaluated. If the adversary passes in a guess of an encoding it gets assigned a random value from the field. Note that here there is no requirement that it is unique.

```plaintext
proc \( O_1(\xi_1, \xi_2) \):
if \( \xi_1 \notin L_1 \) then
    \( f_1 \leftarrow \text{Guess}_1(\xi_1) \)
end if
if \( \xi_2 \notin L_1 \) then
    \( f_2 \leftarrow \text{Guess}_1(\xi_2) \)
end if
get \( f_1 \) and \( f_2 \) from \( L_1 \)
if \( f_3 \in L_1 \) then
    get \( \xi_3 \) from \( L_1 \)
else
    \( \xi_3 \leftarrow \text{Sample}_1(\mathcal{F}) \)
end if
Return \( \xi_3 \)

proc \( O_e(\xi_1, \xi_2) \):
if \( \xi_1 \notin L_1 \) then
    \( f_1 \leftarrow \text{Guess}_1(\xi_1) \)
end if
if \( \xi_2 \notin L_2 \) then
    \( f_2 \leftarrow \text{Guess}_2(\xi_2) \)
end if
get \( f_1 \) from \( L_1 \)
get \( f_2 \) from \( L_2 \)
\( f_3 \leftarrow f_1 \cdot f_2 \)
if \( f_3 \in L_3 \) then
    get \( \xi_3 \) from \( L_3 \)
else
    \( \xi_3 \leftarrow \text{Sample}_3(\mathcal{F}) \)
end if
Return \( \xi_3 \)

proc \( \text{Sample}_1(\mathcal{F}) \):
\( \xi \leftarrow \Xi_1 \setminus L_1 \)
\( L_1 \leftarrow L_1 \cup \{(\mathcal{F}, \xi)\} \)
Return \( \xi \)

proc \( \text{Guess}_1(\xi) \):
\( d \leftarrow Z_p \)
\( L_1 \leftarrow L_1 \cup \{(d, \xi)\} \)
Return \( d \)
```

Figure B.1.: GGM group oracles used within various proofs