UNIVERSITY OF WARSAW
FACULTY OF MATHEMATICS, INFORMATICS AND MECHANICS

Michał Zając
Institute of Informatics

EFFECTIVE CRYPTOGRAPHIC PROTOCOLS WITH LIMITED COMPUTATIONAL POWER AND MEMORY

PhD dissertation

supervisor:
dr hab. Stefan Dziembowski
Institute of Informatics

The author was supported by the National Science Centre grant PRELUDIUM 7 no. UMO-2014/13/N/ST6/03029.

Author's declaration:

aware of legal responsibility I hereby declare that I have written this dissertation myself and all the contents of the dissertation have been obtained by legal means.

Supervisor's declaration:

the dissertation is ready to be reviewed.

Abstract

One of the most important and powerful leakage resiliency model, the Bounded Retrieval Model (BRM) [Dzi06, DCLW06, ADN+10] comes with inevitable space inefficiency caused by a requirement on a cryptographic secret key to be huge, multiple times larger than a leakage bound. Since practical applications demand amount of leakage to be counted in gigabytes, the secret key has to be that large as well. A cryptographic key is usually a long bitstring taken from a uniform distribution. This bitstring occupies gigabytes of disk space and has no other than cryptographic application whatsoever. Although sacrificing such a big amount of memory for sake of security is acceptable for computers, it is usually too much for mobile devices, like tablets or smartphones.

In this thesis we show how to omit this obstacle by deriving a secure cryptographic key in BRM directly from a random variable picked from a non-uniform distribution. We claim that user's private data can be viewed as such random variable. More precisely, we derive a secure cryptographic key from data the user already store on her device. This approach comes with a great advantage — (almost) no additional space is needed to keep the key. Moreover, we ensure that the data remains private in a very strong sense. That is, even someone equipped with the derived key, can say nothing about the underlying data, except information she learnt through leakage. We claim that the delivered procedure fits many known BRM schemes with no modification to the scheme at all.

We call our primitive *key derivation function* (kdf), exactly as in Dodis et al. [DY13]. However, we point out that kdf-s as proposed in [DY13] allow only to construct a key from non-uniform data, but do not ensure data privacy, what is the major contribution of this thesis.

We show an exemplary instantiation of kdf based on disperser graphs (hence we call our function Disperse). The proposed function has an important additional property, called locality. Since a BRM key is large, BRM primitives usually do not use the entirety of it, but some (randomly) picked parts, cf. [Dzi06]. To assure efficiency of our function, we designed it in a way that allows to compute only some required part of the key. Furthermore, to compute a part of the key, it is enough to use only a part of the disk data.

Having kdf proposed, we show some of its applications. We deliver identification and signature schemes based on Merkle trees. Although constructions based on Merkle trees are well known, we show that they remain secure even under leakage. We point out here that the proposed schemes have to fulfill two requirements. Firstly, they have to be asymmetric. In symmetric primitives, all parties participating in the protocol have to share with each other secret keys. Since BRM keys are a few gigabytes large, that would mean a huge disk-space inefficiency. Secondly, even if we use asymmetric scheme, we have to guarantee that public keys are short compared to secret keys. Otherwise, the problem with disk space arises again, because the parties has to keep each others large public keys. We show in this work that the proposed schemes fulfill both requirements.

Part of this thesis (definition of Disperse and its properties) is based on article *Bounded-Retrieval Model with Keys Derived from Private Data* [DDK+16] by Konrad Durnoga,

Stefan Dziembowski, Tomasz Kazana, Michał Zając, and Maciej Zdanowicz.

Key words: cryptography, leakage-resilient cryptography, key derivation schemes, key evolution, time-memory trade-off.

AMS Classification: 94A60, 68P25.

Streszczenie

Jednym z najbardziej rozpowszechnionych i zarazem najmocniejszych modeli w kryptografii odpornej na wycieki jest tzw. *Bounded Retrieval Model* (BRM) [Dzi06, DCLW06, ADN+10]. Niestety, zastosowanie BRM wymaga poświęcenia dużej przestrzeni dyskowej na samo przechowywanie klucza kryptograficznego. Ze względów bezpieczeństwa, tajny klucz w tym modelu musi by większy niż przyjęty parametr wycieku (tj. górne ograniczenie ilości informacji jakie przeciwnik może pozyskać na temat klucza). Chcąc zastosować model BRM w praktyce, parametr ten musi być rzędu kilku gigabajtów. Zazwyczaj klucz kryptograficzny to losowy ciąg binarny, który nie ma innego, poza kryptografią, zastosowania. Oczywiście, można stwierdzić, że przeznaczenie kilku gigabajtów przestrzeni dyskowej w celu zapewnienia bezpieczeństwa nie stanowi większego problemu dla współczesnych komputerów. Z drugiej jednak strony, kilka gigabajtów wolnej przestrzeni jest ilością bardzo dużą dla urządzeń mobilnych, takich jak smartfony czy tablety.

W tej pracy pokazujemy jak można obejść tę przeszkodę poprzez wygenerowanie bezpiecznego klucza BRM-owego ze zmiennej losowej o niejednostajnym rozkładzie. Pokazujemy również, że taką zmienną losową mogą być dane przechowywane przez użytkownika urządzenia. W pracy pokazujemy jak pozyskać bezpieczny klucz kryptograficzny (w modelu BRM) z danych użytkownika takich jak zdjęcia z wakacji, nagrane filmy, czy przechowywane dokumenty. Takie podejście ma zasadniczą zaletę – nie musimy już przeznaczać gigabajtów przestrzeni dyskowej na klucz kryptograficzny, możemy go wygenerować z tego, co i tak przechowujemy na dysku. Przedstawiona w pracy procedura pozyskiwania klucza zapewnia prywatność danych w bardzo mocnym sensie. Pokazujemy, że nawet ktoś, kto zna cały klucz, nie może powiedzieć nic o danych, które posłużyły do jego wygenerowania (poza informacjami pozyskanymi w ramach wycieku).

Ponadto, zaproponowana procedura działa dla wielu schematów w modelu BRM poprzez zwykłe podmienienie jednostajnego klucza na klucz pozyskany za pomocą kdf. Oczywiście, klucz pozyskany z danych niejednostajnych nie jest tak bezpieczny jak klucz pozyskany z rozkładu jednostajnego. Pokazujemy jednak, że strata bezpieczeństwa jest zaniedbywalna (w stosunku do parametru bezpieczeństwa).

Podobnie jak w [DY13], nazywamy naszą procedurę *key derivation function* (kdf). W odróżnieniu jednak od [DY13], zapewniamy nie tylko bezpieczny klucz kryptograficzny z niejednostajnej zmiennej losowej, ale także prywatność danych służących do jego wygenerowania. Możliwość generowania klucza z danych prywatnych jest zasadniczym wkładem przedstawianej pracy.

W pracy pokazujemy konkretny przykład kdf oparty na grafach tzw. disperserach (w związku z tym nazywamy naszą funkcję Disperse). Zaproponowana funkcja posiada bardzo istotną własność, mianowicie jest *lokalna*. Protokoły kryptograficzne w modelu BRM zwykle nie używają całego dostępnego klucza. Ponieważ jest on bardzo duży, próba operowania na jego całości byłaby skazana na całkowitą porażkę z punktu widzenia wydajności. W zamian, za każdym razem, kiedy klucz jest potrzebny, losowana jest pewna jego część, która zostaje następnie użyta. W celu zapewnienia funkcji Disperse

wydajności, zaprojektowana została ona w sposób, który pozwala na obliczenie tylko wybranego fragmentu klucza, bez potrzeby generowania go w całości. Ponadto, obliczenie części klucza nie wymaga dostępu do całości danych dyskowych a tylko pewnego ich fragmentu.

Po zaproponowaniu przykładu funkcji kdf, pokazujemy pewne jej zastosowania praktyczne. Pokazujemy protokoły uwierzytelniania i podpisu cyfrowego oparte na tzw. drzewach Merkla. Choć protokoły te są znane od dłuższego czasu, dowodzimy, że są one bezpieczne nawet w obecności wycieku. Zaznaczmy, protokoły, które nadają się do stosowania w modelu BRM z funkcją kdf muszą spełnić pewne warunki. Po pierwsze, protokół musi działać w modelu kryptografii asymetrycznej. W przypadku protokołu symetrycznego, każda z osób biorąca udział w obliczeniach musi współdzielić prywatny klucz z każdą inną osobą. Ponieważ prywatne klucze w modelu BRM mają wielkość kilku gigabajtów, oznacza to, że każda ze stron musi przechowywać wiele kilkugigabajtowych kluczy. Niewydajność pamięciowa takiego rozwiązania skazuje go na porażkę. Po drugie, w przypadku kryptografii klucza publicznego, klucz publiczny musi być istotnie mniejszy od klucza prywatnego. Wynika to z tego, że w protokole asymetrycznym każda ze stron biorąca udział w obliczeniach musi przechowywać wszystkie klucze publiczne. Jeśli są one duże, schemat jest pamięciowo niewydajny i skazany na porażkę. W pracy pokazujemy, że zaproponowane schematy zapewniają obie własności.

Część rozprawy (definicja i własności funkcji Disperse) powstała na podstawie pracy autorstwa Konrada Durnogi, Stefana Dziembowskiego, Tomasza Kazany, Michała Zająca i Macieja Zdanowicza *Bounded-Retrieval Model with Keys Derived from Private Data* [DDK+16].

Słowa kluczowe: kryptografia, kryptografia odporna na wycieki, schematy tworzenia klucza, ewolucja klucza, złożoność czasowa a pamięciowa.

Klasyfikacja AMS: 94A60, 68P25.

Podziękowania Dziękuję wszystkim tym, dzięki którym powstała ta praca. W pierwszej kolejności mojemu promotorowi dr. hab. Stefanowi Dziembowskiemu i moim współautorom: Konradowi Durnodze, Tomaszowi Kazanie i Maciejowi Emilianowi Zdanowiczowi. Szczególne podziękowania składam także Luizie Zając i Filipowi Mazowieckiemu za długie dyskusje o sensie życia i przychodzenie w sukurs. Dziękuję ponadto Helgerowi Lipmaa za okazane wsparcie i możliwość współpracy z nim w Tartu.

CONTENTS

1. Introduction **13**

1.1. A way to modern cryptography 13
1.2. Security as a game 14
1.3. Motivation for leakage-resilient cryptography 16
1.3.1. Side-channel attacks 16
1.3.2. Modeling side-channel attacks 19
1.3.3. Bounded Retrieval Model 20
1.4. Results 22
1.4.1. Security from private data 22
1.4.2. Overcoming weak expectations 25
1.4.3. Key Derivation Functions (kdf) 26
1.4.4. Disperse as an example of kdf 28
1.4.5. Identification and signature scheme on kdf 29
1.4.6. Key refreshing 30
1.5. Trivial solutions 30
1.6. Organization of the thesis 31

2. Preliminaries **33**

2.1. Security games 33
2.2. Real world vs ideal world 36
2.3. Random Oracle Model 38
2.4. Leakage-resilient cryptography 39
2.5. Various notions of entropy 40
2.6. Identification scheme 43
2.6.1. Making identification non-interactive, Fiat-Shamir paradigm 44
2.7. Signature schemes 44
2.8. From identification schemes to signature schemes 45
2.9. Identification and signature schemes in the Bounded Retrieval Model 46
2.10. Bounded number of executions 48
2.11. Basic properties of disperser graphs 49

3. Disperse as a Key Derivation Function **55**
3.1. Key Derivation Function (kdf) . . . 55
3.1.1. Privacy of key derivation functions . . . 55
3.1.2. Security of key derivation functions . . . 57
3.2. Disperse graph . . . 58
3.3. Guessing game . . . 60
3.4. One-wayness of Disperse . . . 62
3.5. Privacy of Disperse . . . 65
3.6. Security of Disperse . . . 68
3.7. Efficiency of Disperse . . . 72
3.8. Determining real life parameters . . . 73

4. kdf in identification and signature schemes **77**
4.1. Identification in the Bounded Retrieval Model . . . 77
4.2. Construction overview . . . 79
4.3. Merkle tree . . . 80
4.4. Identification on a Merkle tree . . . 82
4.5. Non-interactive identification based on a Merkle-tree . . . 90
4.6. Efficiency . . . 90
4.7. Signature scheme from kdf and a Merkle tree . . . 92
4.8. Public key updates . . . 92

5. Open problems **97**

A. Additional proofs **109**
A.1. Theorem 5.1 from [ADW09] . . . 109

LIST OF FIGURES

1.1. Adversary divided into two parts. 21
1.2. Cryptographic key derivation from private data. 23
1.3. An exemplary $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D)$ built on a 3-regular right disperser $\mathcal{G}_\sigma$. . . . 29

2.1. Execution procedure 34
2.2. ID_n - identification security game 44
2.3. Non-interactive Fiat-Shamir proof system compared to 3-message public-coin proof of argument. 45
2.4. ExUG_n; unforgeability game for signature scheme 46
2.5. $\mathsf{ID}_{\lambda,n}$; leakage-resilient identification scheme security game 48
2.6. $\mathsf{EUG}_{\lambda,n}$; leakage-resilient signature scheme security game 48
2.7. $\mathsf{ID}_{\lambda,n,\zeta(n)}$; $\zeta(n)$-bounded leakage-resilient identification scheme security game. 49
2.8. $\mathsf{EUG}_{\lambda,n,\zeta(n)}$, $\zeta(n)$-bounded leakage-resilient signature scheme security game 50

3.1. Intuitions behind the definition of privacy. 57
3.2. Implementation of $\mathsf{Disperse}$ function. 59
3.3. An exemplary $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D)$ built on a 3-regular right disperser $\mathcal{G}_\sigma$. . . . 60
3.4. Definition of $\mathsf{Guessing}$ game 61
3.5. One-wayness of $\mathsf{Disperse}$. Implementation of a player $\mathcal{P}_\mathcal{A}$. 64
3.6. Privacy. Implementation of the simulator. 67
3.7. Implementation of $\mathcal{A}$ 69
3.8. Intuitions for the proof of Theorem 13 70
3.9. Leakage 1 GB, 256-bit long output of a hash function 75
3.10. Leakage 1 GB, 512-bit long output of a hash function 75
3.11. Leakage 2 GB, 256-bit long output of a hash function 76
3.12. Leakage 2 GB, 512-bit long output of a hash function 76

4.1. An exemplary full Merkle tree 81
4.2. Building a Merkle tree from data blocks 81
4.3. Algorithm $\mathsf{getPath}$ obtains a path from a leaf of index k up to the root c of tree $\mathcal{T}(\ell)$. 83
4.4. An exemplary identification path on a Merkle tree. 83

4.5. Algorithm checkPath checks whether the given path p is a correct path from the leaf of index k up to the root c. 84
4.6. Merkle tree identification protocol 85
4.7. Exemplary execution of a few rounds of the identification protocol. 87
4.8. Adversary $\mathcal{A}$ needs to find such a value of v_l, v_r that $\mathcal{H}(v_l, v_r) = v$. By the assumption, v_r and v are known to $\mathcal{A}$. 88
4.9. Exemplary non-interactive Merkle tree identification protocol 91
4.10. Key update procedure 94
4.11. Implementation of the modified Disperse function. 95

A.1. Reduction from a Σ-adversary to a Π-adversary. 110

CHAPTER 1

INTRODUCTION

1.1. A way to modern cryptography

The problem of private messages exchange bothers people from ages and it can be traced back up to ancient Greeks and Romans. Herodotus in *Histories* [HerBC] tells two stories how Greeks used primitive yet effective steganography. However, the first widely used cipher is known due to the conquerors of Greece, Romans. The so-called Caesar cipher is a shift cipher where every plaintext letter is substituted by a letter lying a fixed number of positions apart (cyclically). This fixed number is called a *key* for the scheme.

There is an important connection between the Caesar cipher and modern cryptographic schemes. Both are based on a rule formulated in the late XIX$^{\text{th}}$ century by a Dutch cryptographer Auguste Kerckhoffs in [Ker83]. This rule, henceforth called *the Kerckhoffs' principle*, states that the security of the system cannot rely on secrecy of the scheme itself (known as *security by obscurity*), which should remain public, but on some secret key.[1] Under this principle we could say that the Caesar cipher is modern. There is no hidden scheme involved, the only secret is the key.

Kerckhoffs' principle lies underneath the most popular cryptographic schemes which are used world-widely every day by billions of people, like introduced in the 1978 RSA [RSA78], developed in 1985 El Gamal [EG85], announced in 1997 AES [Nat01], and others. Both RSA and El Gamal security has been proven by a reduction from a computationally hard problem. The former introduced so-called RSA problem, which can be solved if one can efficiently factorize complex numbers, but no other general method is known (moreover, it has been shown that in the generic group model factorizing and solving RSA problem can be reduced to each other [AM16]). For the latter, one can show a reduction from the

[1] We should emphasize here that probably many military used cryptographic tools rely on both secrecy of the scheme and the key.

discrete logarithm problem, which (along with its multiple variations) is one of the most commonly used problems to show the security of a cryptographic scheme.

Although almost every secure Internet connection uses at least one of the aforementioned schemes, there are usually no proofs that any of them remains secure when the key is not uniform (i.e. taken from a non-uniform distribution) or when a malicious adversary learned some information about it, see e.g. [NSS+17]. Furthermore, the most famous asymmetric encryption scheme, i.e. RSA, in one of its variation that has been proposed to improve efficiency of a scheme, is absolutely insecure if a malicious party focused on breaking the scheme can introduce a slight error in computations [GGOQ98]. We will address the problem of key non-uniformity later.

1.2. Security as a game

We used above several times words like *secure* and *break*, here we introduce informally some hints about their meaning. Usually, modern cryptography proves the security of a scheme by defining a game, here called a *security game* (for formalization of a security game see Section 2.1), played between a malicious party, called adversary $\mathcal{A}$ and another party, called challenger, that answers $\mathcal{A}$'s questions and requests. For example in identification schemes where $\mathcal{A}$ tries to impersonate a legitimate user, prover $\mathcal{P}$, communication goes between adversary $\mathcal{A}$ and verifier $\mathcal{V}$ that verifies whether a party she is talking to is who the party claims to be[2]. In another case, this counterparty may be an oracle that encrypts plaintexts chosen by the other party, etc. In this work we define a number of security games, usually between the adversary and the verifier (e.g. see Figures 2.2, 2.4, 2.5 and 2.6).

To give an illustration how security games look like, we provide a simple example of a game used to prove that an encryption scheme is secure. Encryption scheme is a triple of algorithms: $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$. First, KeyGen takes as input security parameter n written unary (denoted by 1^n) and randomness R, then outputs a key k from some distribution K. Latter algorithms, Enc and Dec, handle encryption and decryption operations. Algorithm Enc is probabilistic, while Dec is deterministic. For message m drawn from a set of all possible messages **MsgSpace** we have $m = \mathsf{Dec}_k(\mathsf{Enc}_k(m))$.[3]

The security of an encryption scheme is often defined as follows: the adversary, even equipped with an access to the encrypting oracle, learns nothing about the plaintext. Hence for every random variable M over **MsgSpace**, M and $\mathsf{Enc}_K(M)$ are independent from the adversary's point of view. We say that an encryption scheme is *perfectly secure* if, M and $\mathsf{Enc}_K(M)$ are just independent random variables. However, we usually require[4] that they only *look* independent for some PPT (see Definition 4) adversary $\mathcal{A}$.

[2]Usually, the verifier is not a party that answers questions, but she responds on identification requests with challenges.

[3]See, this description fits also to asymmetric encryption scheme, in such $m = \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m))$.

[4]Perfectly secure encryption schemes are quite inconvenient since they require secret key of length equal messages.

This property can be formalized by a game described below. There are two parties, adversary $\mathcal{A}$ and encryption oracle $\mathcal{E}$. The latter, responds on adversary's query m with $\mathsf{Enc}_k(m)$ for some k picked uniformly random. The number of queries the adversary can submit is bounded by some polynomial function $poly(n)$, where n is a security parameter.

Remark. In this thesis we often use expressions like *polynomially many* (e.g. polynomially many queries to an oracle, see Example 1 below) or *negligibly* (e.g. negligible function, see explanation below Example 1). In all these cases we think about functions in the *security parameter* n. That is, the adversary which asks polynomially many queries can ask up to $poly(n)$ queries for some polynomial function $poly$ and security parameter n. Similarly, in the other case, function ε is negligible for security parameter n.

Example 1 (Security game for an encryption scheme)**.** The game goes as follows:

1. **Setup stage:** Let $\mathsf{Enc}, \mathsf{Dec}$ be encryption, decryption algorithms. Key k comes from KeyGen run on 1^n for security parameter n and uniformly random R. By $\mathcal{E}^k$ we denote the encryption oracle, which on input m responds with $\mathsf{Enc}_k(m)$. Both $\mathcal{E}^k$ and adversary $\mathcal{A}$ are given k and other public parameters of the scheme.

2. **Learning stage:** Adversary $\mathcal{A}$ submits polynomially many messages m_i and gets answers $\mathsf{Enc}_k(m_i)$ from $\mathcal{E}^k$. $\mathcal{A}$ can pick her messages adaptively, that is, decide on m_i after seeing $(m_j, \mathsf{Enc}_k(m_j))$, for $j < i$.

3. **Test stage:** Next, $\mathcal{A}$ chooses a pair of messages m_0, m_1 and sends it to $\mathcal{E}^k$. Oracle $\mathcal{E}^k$ chooses at random bit b and responses with $\mathsf{Enc}_k(m_b)$. This is followed by adversary $\mathcal{A}$ that outputs bit b'.

We say that the adversary wins the game if $b = b'$.

Remark. The example above works also for asymmetric encryption schemes. In the asymmetric encryption scheme KeyGen produces a pair of keys $(\mathtt{pk}, \mathtt{sk})$. The former is called *public key* and used for encryption, the latter is called *secret key* and used to decrypt messages. To adjust the example above to an asymmetric encryption scheme we should parametrize the encrypting oracle $\mathcal{E}$ by a public key $\mathtt{pk}$ not secret key k.

We say that an encryption scheme is *secure*[5] if the adversary wins the game (i.e. she *breaks* the security) described in Example 1 with only negligible (we say that a function $f : \mathbb{N} \to [0, 1]$ is negligible iff $f(x) = O(x^{-c})$ for every constant $c > 0$, see Definition 3) probability over $\frac{1}{2}$. Note that the adversary can always guess randomly, in such case she guesses correct bit with probability 1/2.

[5]This notion of security of an encryption scheme is called CPA-security, where CPA stands for Chosen Plaintext Attack.

As said before, many security proofs do not hold if the key is not uniformly random or the adversary has some additional knowledge about it (usually information like key length is public, unlike information on number of '1' in it or value of a particular bit). Thus, the discussion on the crucial assumption of key randomness uniformity and key secrecy is a must. This leads us directly to the next part of this thesis.

1.3. Motivation for leakage-resilient cryptography

In this part we explain why the assumption of the unreservedly secret private key may be a little too optimistic. Moreover, we present main ideas on preventing the adversary from breaking schemes while the above-mentioned assumption does not hold and she can get some side information on the secret key. In such case, we say that the adversary *leaks* information or that there is *leakage*, see Section 2.4 for formal definitions of leakage-resilient cryptography.

1.3.1. Side-channel attacks

Although a proof of the security for a cryptographic scheme may be perfectly correct on the paper, it may not fit well to the real-world applications. Main problems arise because of three factors: the scheme is *implemented* and may contain implementation failures, it is *run* on some operation system, which can be attacked by malicious software, and it is *executed* on a hardware device. Faulty implementations, radiation from electronics, memory leaks make some side-channel information about secrets processed on a device achievable for a malicious user. Here we mention just a few issues that have to be considered while using cryptography:

Malicious software A group of possible attacks comes with the omnipresence of the Internet. Almost every modern electronic appliance is connected to the network inevitably being exposed to thousands of viruses, trojans and others. Once a device is infected with such malicious software or has been taken over by a hacker, we cannot preserve faith in effectiveness of cryptography run therein. Here we present two simple, yet based on the real life, examples of how malicious software can cause a security risk.

> **Example 2** (Malicious software and cryptography [SJB+14])**.** Assume that a virus $\mathcal{A}_{\text{virus}}$, created by some malicious $\mathcal{A}_{\text{creator}}$ and residing on a machine $\mathcal{C}$, has access to any part of device's memory. Especially these parts where cryptographic keys are stored. Then $\mathcal{A}_{\text{virus}}$ can do the following:
>
> 1. find a part of $\mathcal{C}$'s memory that contains a secret key k used to authenticate bank transfers;
>
> 2. connect to a bank server using k;

3. request a transfer of all funds directly to an account managed by $\mathcal{A}_{\text{creator}}$ [6].

In the Example 2 any kind of cryptography is useless. If a virus or a hacker are present, they can proceed on behalf of the legitimate user and learn all user's cryptographic keys.

This example is motivated by a real-life attack performed by Silver et al. [SJB+14]. The researchers have shown a method to retrieve passwords stored in a password wallet by manipulating user's browser to navigate, without notifying the user, to a malicious webpage.

The second example, Example 3, is a little different. We assume that the user does not store keys and passwords locally, but inputs them every time when it is necessary. Thus, there is no place on a disk where secrets are stored permanently. However, since software that records every keyboard stroke exists (so-called key-loggers), we show that this countermeasure can be circumvented as well [Spr16].

Example 3 (Malicious software and cryptography (2) [Spr16])**.** We assume that virus $\mathcal{A}_{\text{virus}}$, created by some $\mathcal{A}_{\text{creator}}$ and residing on machine $\mathcal{C}$ is able to record keystrokes input by the legitimate machine user. Then $\mathcal{A}_{\text{virus}}$ can do the following:

1. wait till the legitimate user wants to log into a bank account;
2. during logging, record keys pressed retrieving user's login `login` and password `password`,
3. upload a (`login`, `password`)-pair to a bulletin board.

Then creator $\mathcal{A}_{\text{creator}}$ finds both `login` and `password` and is able to identify to a bank on behalf of the legitimate user.

However, in the presented examples we can run some cryptographic countermeasures if we only assume that the hacker or the virus can upload some limited amount of information and from time to time the machine is virus-free. The idea for defence is simple – make sensitive information so huge that no one can upload them unsuspiciously. This approach will be elaborated later.

Attacks on hardware The next source of threats is a device which runs cryptographic schemes. Real-world scenarios forces us to assume that the adversary has physical access to the mentioned device. Hence, we have to take into consideration that some additional information may be leaked, e.g. by measuring such processes like:

electro-magnetic radiation Modern cryptographic hardware uses electric current. Thus, they emit electro-magnetic radiation that can be measured by a number of more

[6] In this scenario we do not care whether the perpetrator is easily identifiable or not; we can assume that even if this is the case, she is out of our reach.

or less sophisticated probes. These measurements may reveal information about underlying computation since the radiation reveals information about the characteristics of the current. Countermeasures taken to prevent the adversary from probing radiation may be based on e.g. shielding, i.e. covering the device by a layer of radiation-absorbing material or putting defended machine into a Faraday cage. Another countermeasure is to design the circuit in such a way that the radiation of separate wires cancel each other (so-called twisted-pair cables invented by Bell [Bel81]). Although these countermeasures may be efficient against some adversaries, they are usually very expensive and cannot be used on a wide scale. Below we show a real-life example of how radiation measurements affect the security.

Example 4 (Van Eck phreaking)**.** Van Eck showed that using a $100 apparatus one can eavesdrop CRT display from hundreds of meters [Hig87]. Although the attack was presented in 1988, it was known to the military before. Even though CRT displays are no longer popular, the same attack works on LCD screens.

computation time A seminal paper by Kocher [Koc96] drew attention to another problem with security hardware and implementations. The paper shows that a naïve implementation of systems like RSA or Diffie-Hellmann can be vulnerable because of differences in computing time depending on a value of the private key.

Example 5 (Computation time of RSA)**.** It was shown in [Koc96] that if a number of RSA ciphertexts can be submitted to decryption, an attacker can reveal the secret key completely by observing only computation time.

memory probing In this kind of attack we assume that there is a way to obtain values of some bits stored in a device's memory. Out of plethora of memory attacks we give one as an example.

Example 6 (Halderman et al. attack)**.** Halderman et al. [HSH+08] shown that it is feasible to retrieve a considerable part of computer's DRAM memory (or other volatile memory) by powering it off and freezing the device. Such a procedure allows to read information directly from the DRAM unit, what can reveal the secret key if it has been transferred to the volatile memory during computation like decrypting, signing, etc.

To conclude, a huge amount of side-channel attacks are possible due to electromagnetic and heating radiation or computation time measurements. Such vulnerabilities can be

limited by, e.g. redesigning the hardware to reduce radiation. However, that usually demands more materials used and more complicated circuits. Furthermore, a device can be stored in a special box, which reduce the possibility of leakage even more. This approach make hardware much more expensive and inevitably much more complicated and harder to analyze. Thus, these solutions are not widespread but used only by users like military forces. See [Sta10, BMV05] for detailed information on hardware side-channel attacks and countermeasures.

Implementation failures Using low-level programming language like C comes with potential security dangers caused by the huge flexibility in managing memory of a device (see [CW07, HL02, MM05]). This feature forces a programmer to consider attacks like buffer overflow and similar. Just to mention a recent example, the Heartbleed attack [Cod14, Sch14], which allowed a malicious user to obtain a secret key used in standardised X.509 certificates from a TLS/SSL session. This attack was particularly devastating since the TLS/SSL protocol was supposed to secure internet connection and is world-widely used in online banking and shopping. The heartbleed bug can be described in a simple attack scenario.

Example 7 (Heartbleed)**.** Let $\mathcal{C}$ be a client and $\mathcal{S}$ a server. Suppose that $\mathcal{C}$ checks whether $\mathcal{S}$ is still connected, then:

- If client $\mathcal{C}$ is honest, she asks the server question like: `If you are there, send me these 5 letters: 'BREAD'`.
  Server answers `BREAD`.
- On the other side, if client $\mathcal{C}$ is malicious, she can change a question a bit: `If you are there, send me these 500 letters: 'BREAD'`.
  Server answers `BREAD` and 495 following characters as requested.

$\mathcal{C}$ gets from $\mathcal{S}$ the requested number of letters despite of the length of the word she asked. Along these additional letters malicious client $\mathcal{C}$ may obtain the value of server's password and other sensitive information.

This particular example was possible due to the high flexibility that C gives. Even though many modern high-level languages like Java are free from buffer overflow attacks, we can not assume that programs written in these languages are safe.

1.3.2. Modeling side-channel attacks

Engineering approach to side-channel attacks issues relies on protecting hardware units by techniques like, e.g. aforementioned shielding or implementation modifications, leaving the underlying cryptographic scheme as it was. The main drawback of such an approach

is impossibility of anticipating all possible attacks on a device. A slightly different technique of measuring radiation can make all conceived security useless. Furthermore, more and more accurate probing devices are developed, making protection of sensitive data respectively more difficult.

We employ a different approach to provide the security under side-channel attacks. We model mathematically capabilities of the adversary and design schemes provably secure in the considered model like in [Dzi06, Pie09, FKPR10, DP08, ADN+10, DORS08]. The list below presents some popular approaches on modelling leaky devices and implementations.

Only computation leaks information This model, presented e.g. in [GR15, GR10, DDN15], assumes that the adversary does not have access to the whole secret, but only to a part being currently used in computation. A scheme secure in this model is resilient to the attack described in Example 5.

Memory leakage On the other hand, there is the memory leakage model (see [BKKV10, DLWW11, DHLW10, KKS11]), where the attention is paid not to computation but to memory of a device. Example 6 illustrates an attack covered by this model.

Relative and absolute leakage In these models the adversary has access to the whole device, both memory and parts used in computation. However, information she can obtain is limited by some constant number of bits λ. These models fit especially well to cover attacks described in Example 3. Furthermore, the attacks described in Example 5 and 6 would also be pushed back. This thesis is done according to the absolute leakage model, more precisely the Bounded Retrieval Model. We elaborate more on this particular approach below and in Section 2.4.

1.3.3. Bounded Retrieval Model

In the world of hackers, Internet and malicious software spread therein a powerful model of leakage is strongly demanded. Consider a scenario when a given device is infected by some virus. We should assume that the virus has almost unlimited access to device's disk data and memory. Especially, it is granted access to any cryptographic keys stored therein. In such case, the virus can authenticate bank transfers (cf. Example 2), retrieve passwords (cf. Example 3), decrypt private letters, etc. All of this on behalf of the legitimate user without any obstacles or suspicions. Unfortunately, it is not possible to provide any (provable) security in such case.

On the other hand, we can assume that the virus can be detected and removed by some antivirus software, hacker can log out and so on. We can assume that there is a period of time, when the machine is virus-free. We provide a scheme that is secure in the following sense.

- Divide adversary $\mathcal{A}$ into two instances, internal $\mathcal{A}_{\text{int}}$ with unlimited access to machine's resources and another, external $\mathcal{A}_{\text{ext}}$, that retrieves information from $\mathcal{A}_{\text{int}}$ and makes use of them. We could interpret the pair of the adversaries as follows

– $\mathcal{A}_{\text{int}}$ is malicious software residing on the machine and $\mathcal{A}_{\text{ext}}$ is its creator. See Figure 1.1 for an illustration of such an adversary.

- Define a security game Game (cf. Definition 1) that will be played by $\mathcal{A}_{\text{ext}}$.
- Assume information that $\mathcal{A}_{\text{int}}$ can pass to $\mathcal{A}_{\text{ext}}$ is limited.
- Any information that $\mathcal{A}_{\text{int}}$ can pass does not help $\mathcal{A}_{\text{ext}}$ win Game with non-negligible probability over 0 or $\frac{1}{2}$ (depending on Game).

The assumption on the limited amount of information that $\mathcal{A}_{\text{int}}$ can pass $\mathcal{A}_{\text{ext}}$ can be justified as follows. Let this amount be measured in gigabytes of data, then it is impossible for $\mathcal{A}_{\text{int}}$ to sent such amount of data unsuspiciously. That is, we expect that a user notices such a great leak and stop it by, e.g. disconnecting the machine from the Internet.

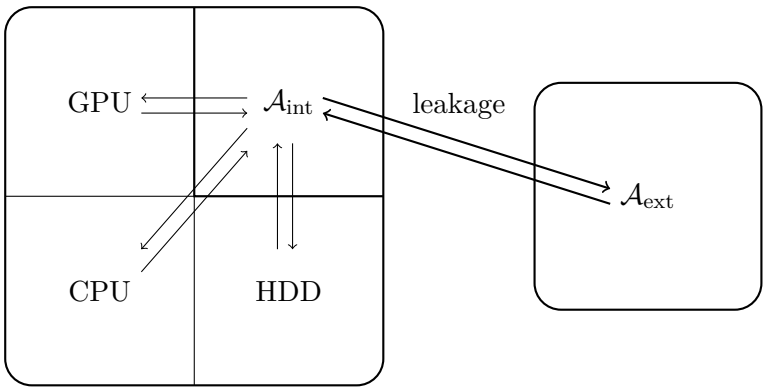


Figure 1.1.: Adversary divided into two parts.

Consider for example a signature scheme. The simplest method to achieve the security of the scheme in the setting presented above is to make any sensitive information huge. Instead of a few kilobytes long keys, which can be uploaded to the Internet in no time, keys could be made huge, a few gigabytes long. Our goal is to show that even if the adversary learn gigabytes of information it does not help her to break the security of the scheme.

This solution, called the Bounded Retrieval Model, has been proposed in parallel in [Dzi06] and [DCLW06] and thereafter developed in many works like [ADN$^+$10, ADW09, DS05, DP08, DKW11, NS12, DP07, FPS12, FKPR10, BKR16], where authors provided cryptographic primitives like encryption, identification, signature and secret sharing secure in this model. Since BRM gives the adversary a great power, this thesis is done according to it.

We model knowledge the adversary can learn by *leakage queries*. These queries can be described as functions of secret data. The adversary submits them to a *leakage oracle* that computes values of these functions taking secrets as an argument. The adversary learns outputs of the functions if they are (cumulatively) shorter than a *leakage parameter*,

which is set to prevent the adversary from learning too much (e.g. the whole secret). See Section 2.4 for more information and precise model description.

Drawbacks of BRM

Despite of obvious advantages of the leakage-resilient cryptography (especially BRM) over the traditional approach to security, there are a few drawbacks of a great importance which cause leakage-resilient cryptography still unpopular and absent in the real-world cryptographic implementations. (However, papers like [FPS12] provides some ideas how to make leakage resilient practical). Here we mention only these, which are relevant to the technical point of view.

Memory inefficiency Since a key in BRM is usually a few *gigabytes* long it is incomparable larger than keys used in RSA or El Gamal schemes which occupy only a few *kilobytes* of device's memory.

Running time increase Memory inefficiency goes along with another drawback. That is, increase of a running time of a scheme. This is caused by either a must of picking successive bits of a cryptographic key from much larger memory space, or inevitable bigger algorithmic complexity of a scheme.

Regarding the first obstacle, users are forced to sacrifice a large amount of memory just to store a giant cryptographic key that is otherwise useless. This makes BRM not practical for mobile devices. Although personal computers have usually great computational power and hardware underneath can be easily upgraded, RAM made faster, HDD space added, smartphones and tablets are much less accessible and less upgradeable devices. In this work we show how to circumvent this obstacle.

The second drawback can be circumvented by using only a randomly chosen part of the key (see [Dzi06]), what limits computation complexity significantly. Instead of running computation on the whole key, considered as a long sequence of random bits, only a relative short subsequence of it is used. This subsequence is chosen randomly for every single use of the key. Assuming that countermeasures against the second obstacle are sufficient, we focus on how to circumvent the first obstacle.

1.4. Results

1.4.1. Security from private data

The main result of this thesis is a key derivation function, which provides a secure cryptographic key in the bounded retrieval model from private, non-uniform data. Obtaining cryptographic keys from non-uniform data is a well-known problem, elaborated in a number of papers, see Section 1.4.2 below for references. The true novelty this thesis brings is use of private data for that purpose. Here we discuss benefits that come with this

approach. We also point obstacles that had to be circumvented to make use of private data possible. Finally, we discuss which data we consider private and which we do not. Figure 1.2 illustrates general idea of a key derivation process from user's data.

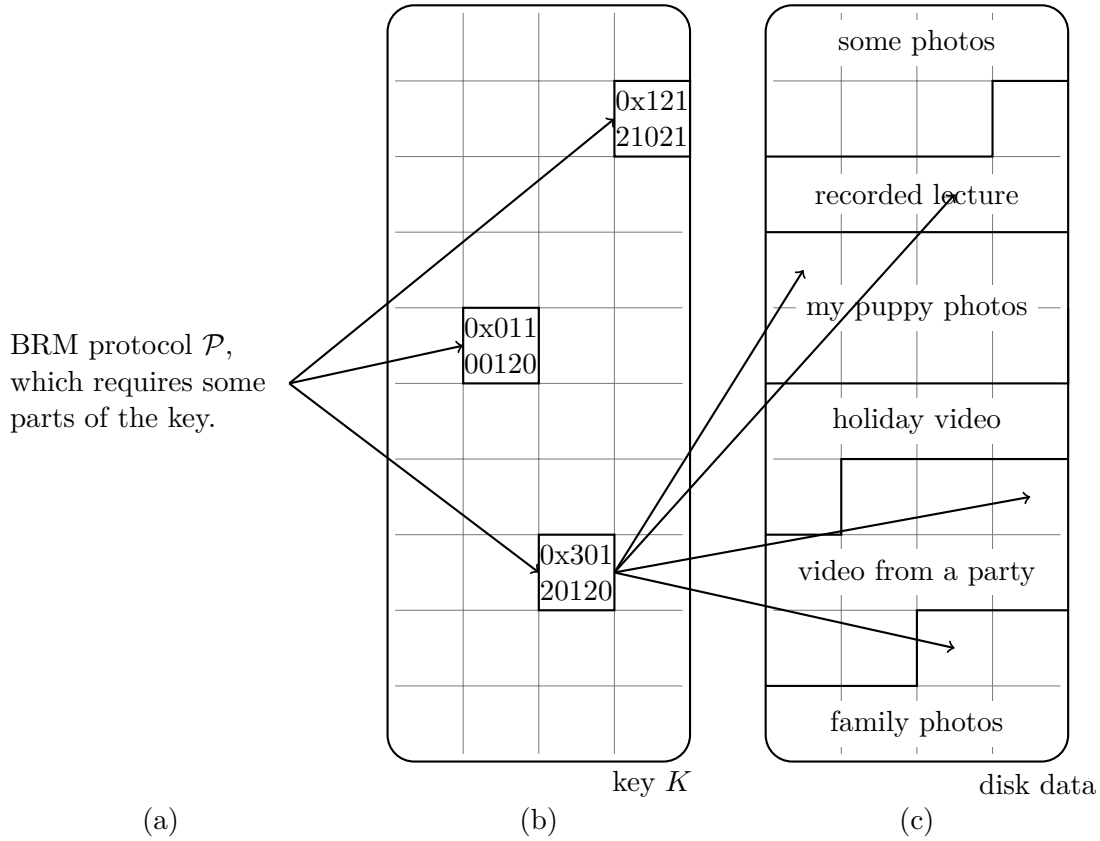


Figure 1.2.: Cryptographic key derivation from private data. Picture (a) shows a BRM protocol that needs to use some parts of a secret key K, cf. Picture (b). The key is derived from disk data, which consist of music, images and videos stored by the user on a disk as shown on Picture (c).

Private data, benefits and obstacles As mentioned before, the novelty of our approach is to employ user's private data to generate a cryptographic key secure in BRM. This makes BRM-based cryptography available for mobile devices, which dispose only limited amount of disk space. Instead of sacrificing pricey memory to store a huge block of random bits that is useful only for cryptographic purposes and otherwise useless, we use data that has been already stored on a device.

Such an approach forces us to assure that the data we use remain private. This is especially important in BRM, which allows an adversary to learn information about the key by queries to a leakage oracle. Furthermore, in this model we (sometimes) allow partial disclosure of the key, e.g. if the key is used to identification, then the verifier could learn some parts of it during the protocol execution. Motivated to make our key derivation

procedure as universal as possible, we have to consider such threats. Here, we employ very conservative approach. We show that the adversary learns nothing (except allowed leakage) about the underlying data even if she is given the whole derived key.

This thesis shows techniques allowing to enhance security of mobile devices considerably and at low cost. The device will remain secure even if the adversary learns parts of the cryptographic key. The cost is a little slower cryptographic operations, since the key parts are generated every time the key is used. On the other hand, we address this drawback also. The user could simply generate some parts of the key before the protocol starts and store them. However, this limits benefits of the proposed solution – stored parts look like random bits and does not have non-cryptographic application. This is a trade-off. The user can decide how much of free space she may utilize to store the key and compute as many parts of the key as she wants. Furthermore, she can change her mind at any time and either precompute more parts, making further cryptographic operations faster, or remove them to obtain more free space.

Which data are private? We have to address a question: which data stored on our devices could be used in key derivation? The answer is not as straight-forward as it may seem. For example, if we store on a device episodes of a beloved TV show, could we use them? Unfortunately, the answer is usually negative here. Suppose that we want to use a certain episode of *Better call Saul* TV series. Then the adversary, who is equipped with access to the leakage oracle could just query it for the title, the number of series and episode. This information are usually not secret and provided, e.g. in file metadata. Furthermore, almost every episode of a (popular) TV-show is recognizable by its hash value[7]. Thus, the adversary easily learns which file was used consuming only few bits of leakage. She could then download the file from the Internet obtaining some non-trivial information about the derived key.

Of course, we could argue that there are many versions of the file on the Internet. However, we take here a conservative stance and claim that the number of versions is too limited to consider using publicly available multimedia files to derive a key.

On the other hand, we believe that some uncertainty of a multimedia file can be preserved in some cases. Suppose that the user stores movies on a PC and before she transfers them to a mobile device she converts them to a mobile-friendly format (e.g. losses some quality, changes file format). Such a conversion, with a great probability, will make recognition of file by its hash value impossible. However, we should still be careful, since some data may be read from metadata. It is also hard to predict how much the converted file would differ from the original one.

Another countermeasure that could be taken to make publicly obtainable files useful to key derivation is personalization. Of course, we cannot demand from the user to, e.g. change colours of a movie, add scenes there or add tracks to a music file. Here, by personalization we mean including in file some data specific and unique to user's device.

[7]This fact is widely used in video players that recognize file and download subtitles tailored for it.

For example, if a certain block of data from a movie is used in key derivation, we could add the number of a disk sector it occupies. We believe that even if the adversary easily (and cheaply, in terms of leakage) learns which movie file was used, it would cost her much more to obtain all disk sector indices also.

Ideally, we would like to assume that multimedia created by the user and stored on a device are not available anywhere else. Unfortunately, in times of social media we cannot do so. Thus, we have to address the following question: what if a file (say, a picture) used in a key derivation procedure was also uploaded to some social media website, like Facebook? Here we argue twofold. Firstly, on sites like Facebook media are converted and compressed during upload, because of limited capacities of servers. Thus, the file which occurs on the site differs from the file from the device. Secondly, we put trust in user's security and privacy routine. We assume that private pictures will not be shared with all of the world but only inside a bounded (however potentially big) group of friends. We claim that both arguments make possibility of obtaining an exact copy of the file stored on the device considerably reduced.

This discussion boils down to a conclusion. Because of security reasons, we argue that files used in a key derivation procedure should not be easily obtainable and identifiable on the Internet. Contrary, they should be produced by the user herself or, at least, personalized by her, e.g. by adding disk-specific information to them as mentioned above.

One could ask whether users produce enough content to even consider deriving cryptographic keys from created data. Here we claim that this is the case. We support this claim by two statistics: every minute 300 hours of movies is uploaded to Youtube [Bra16] and 1.8 billion photos is shared on Facebook, Snapchat, Instagram every day [Eve15]. This shows how creative people are and how much data they produce.

1.4.2. Overcoming weak expectations

Classical cryptographic schemes have the security based on randomness taken from a uniform distribution, which is not easily achievable in the real world. Hence, a lot of effort has been put to deliver schemes which do not depend on such rare and fugitive primitive and retain comparable level of security. This line of research has been proposed by Barak et al. in [BDK$^+$11]. The paper proved renowned Leftover Hash Lemma stating that a family of hash functions constitute good randomness extractors. Research has been continued and developed in e.g. [DY13] and [YL13].

On the other hand, there are plenty of papers on obtaining randomness close to uniform from sources like (a) nature, e.g. from biometric data [DORS08, BDK$^+$05], (b) hardware [BH05, BST03], (c) samples from close distributions [DKK$^+$12], (d) repeating condensing [RSW06], (e) random group element [CFPZ09], (f) different modes of encryption operation [DGH$^+$04].

These solutions work in two different settings. In papers like [BH05, BST03, CFPZ09, DGH$^+$04] authors prove that indistinguishability of a perpetrated and uniform randomness holds only if $\mathcal{A}$ is computationally bounded, i.e. there is some constant T such that

$\mathcal{A}$ cannot perform more than T operations (e.g. in terms of Turing machines, no more than T transitions). On the other hand, [DORS08, BDK+05, DKK+12, RSW06] do not make use of that assumption and provide results based on information theory. This thesis employs the information-theory setting. However, results obtained here work for computationally bounded adversaries also.[8]

1.4.3. Key Derivation Functions (kdf)

Key Derivation Functions are where above-mentioned problems of BRM space inefficiency and derivation of cryptographic key from non-uniform source of randomness meet. A number of authors proposed solutions for both of them, especially for the latter one. Up to our knowledge, we are the first who produce efficiently computable BRM key from imperfect sources of randomness like private data already stored on a device. Provided solution is secure and private, it does not give an adversary any (non-negligible) additional power and preserves privacy of used data.

The idea of Key Derivation Functions comes directly from [DY13], where the authors have shown a concept of a new class of functions kdf that for source $D \in \{0,1\}^m$ of min-entropy at least κ (see Definition 10 for definition) outputs a cryptographically secure secret key $K \in \{0,1\}^n$. Here we enhance this concept by showing how to use kdf in BRM. We also define security of kdf in this model. Moreover, due to our application and fact that a min-entropy source D may be sensible information itself, we introduce a concept of private kdf, which hides information about D. Last but not least, we provide a working example of kdf which is described later.

Key derivation vs randomness derivation BRM schemes usually consists of a key generation subroutine KeyGen that on input $(1^n, R)$, where n is a security parameter and R is a random variable of uniform distribution, produces a secret key for the scheme. Our kdf could be used to produce *fake* randomness R' that is not uniformly distributed, yet has some min-entropy (for definition of min-entropy check Definition 10). That is, for given non-uniform D, we run KeyGen procedure on some $R' \leftarrow \mathsf{kdf}(n, D)$. More precisely, the key for the scheme is produced by $\mathsf{KeyGen}(1^n, \mathsf{kdf}(n, D))$.

However, in most cases we will consider really simple KeyGen functions that on input $(1^n, R)$, return just R. That is the reason why we call our primitive *key derivation function*, not *randomness derivation function*.

However, asymmetric cryptography requires a key divided into two parts – one public $\mathtt{pk}$ and another private $\mathtt{sk}$. In such setting we set $\mathtt{sk} = R$ and compute $\mathtt{pk}$ as a deterministic function on $\mathtt{sk}$. When referring to *key*, we talk about $\mathtt{sk}$.

Having this remark in mind, we later write $\mathsf{KeyGen}(1^n, R)$ if the uniformly random key is generated given uniformly random R and $\mathsf{kdf}(1^n, D)$ if the key is derived by kdf on some

[8] Note, it is not straight-forward that information-theoretic results work for computationally bounded adversaries also. In principle in the former setting proofs may use techniques unachievable for proofs in the latter setting, like computationally unbounded simulators.

D that may not have full min-entropy.

Security of kdf The main property of kdf is security meant as a gap between the security of a scheme working under uniformly random key and the same scheme where the key was derived by kdf.

Suppose that a BRM scheme Prot is $\varepsilon(n)$-secure in terms of the corresponding security game Game (see Definition 1). We assume that Prot contains as a subroutine function KeyGen that given security parameter n and access to randomness R, that is uniformly distributed, returns a key K. Consider another scheme Prot$'$ that differs to Prot in one detail only – while Prot uses KeyGen, Prot$'$ produces the key using kdf on randomness D, provided that D has high min-entropy. We say that kdf is $\varepsilon'(n)$-secure if scheme Prot$'$ is $(\varepsilon(n) + \varepsilon'(n))$-secure.

Example 8 contains simple yet illustrative example of this property. We show how security of kdf affects the security of an encryption scheme. Delivered solution has a minor overhead on the amount of side-information the adversary obtains. To keep example simple, we did not equip the adversary with access to a leakage oracle.

> **Example 8** (Encryption scheme with a key derived from kdf)**.** Recall a game described in Example 1. We say that an encryption scheme is $\varepsilon(n)$-secure if the adversary wins a game described in Example 1 with probability $1/2+\varepsilon(n)$, for some negligible ε. In the example, we assumed that a pair of keys $(\mathtt{pk}, \mathtt{sk})$ comes from a KeyGen function run on a uniformly random R and 1^n. More precisely, sk is uniformly random and pk is obtained deterministically given sk.
>
> Here, instead of a KeyGen subroutine, we use kdf run on a random variable D. The scheme that obtains sk from $\mathsf{kdf}(1^n, D)$ is $(\varepsilon(n) + \varepsilon'(n))$-secure in terms of a security game described in Example 1.

What is crucial in this definition is that it fits to any BRM scheme with the security defined by a game and key generation procedure that can be run by the party independently.[9] It does not redefine the security for a scheme, but describes the security gap between a uniform and a non-uniform key.

Privacy of kdf Using private data to derive cryptographic key in BRM comes with additional vulnerability. Many BRM protocols allow partial disclosure of the key during identification, signing, encrypting and other cryptographic operations. Furthermore, some of them assume that all parties are given access to it. All of this call under question the idea of using private data. The proposed solution assures that private data remains private.

The idea of the definition of privacy goes as follows. Suppose that algorithm $\mathcal{A}$ on input K derived from private data D (where private data D stands for a realization of a random

[9] As an example for a BRM protocol that does not this property recall [ADW09], where the users of the protocol require *master update key* that is common for all of them and is not prone to leakage.

variable with distribution corresponding to the distribution of disk data) retrieves some information about D. We construct a machine called the simulator (see Section 2.2 for definition and intuitions on simulators and their role in the provable security), denoted by $\mathcal{S}$, which gets $\mathcal{A}$ as input, i.e. $\mathcal{A}$ is described as a Turing machine and given to $\mathcal{S}$. However, $\mathcal{S}$ is not given access to the key K or other $\mathcal{A}$ inputs. We say that data D remain private if the output of $\mathcal{S}$ on $\mathcal{A}$ has distribution indistinguishable from the distribution of the output of $\mathcal{A}$ with K on the input, even conditioned on particular value of data D.

An exemplary measure of the indistinguishability of two distributions is a statistical distance (denoted by Δ), defined as follows:

$$\Delta(X,Y) = \frac{1}{2} \sum_{a \in \mathrm{supp}(X) \cup \mathrm{supp}(Y)} | \Pr(X = a) - \Pr(Y = a) | \ .$$

Indistinguishability of the aforementioned distributions can be formally written as:

$$\Delta((\mathbf{Output}(\mathcal{A}(K)), D), (\mathbf{Output}(\mathcal{S}(\mathcal{A})), D)) \leqslant \varepsilon(n) \ ,$$

for negligible ε. We write **Output** to emphasize that we consider only outputs of $\mathcal{A}$ and $\mathcal{S}$, and we do not compare distributions of algorithms.

The conclusion from the presented formula is following: since the output of any algorithm $\mathcal{A}$ can be simulated by an algorithm with no access to K, no algorithm $\mathcal{A}$ obtains any non-trivial information on D from K. However, it does not mean that the adversary remains absolutely oblivious to data D. Since she is equipped with leakage, she can obtain information using it. However, that is the only way she can learn something about data D.

Note that this property allows us to use a key derived by kdf also in symmetric BRM schemes. More precisely, since the key reveals nothing about the underlying data a party that derived it can share it with another party, without putting privacy of data at risk.

1.4.4. Disperse as an example of kdf

As an example of a key derivation function we describe a function Disperse that transforms any input D to a secure and private BRM key, where parameters for security and privacy depends on the allowed amount of adversarial leakage and min-entropy of D. Function Disperse is built on a d-regular disperser graph $\mathcal{G}_\sigma$ (see Section 2.11 for more information about disperser graph) along with a random oracle $\mathcal{H}$ (see Section 2.3 for more information about random oracles and how they can be instantiated in practice by hash functions). Exemplary constructions are shown on Figure 3.3. Here D_i stands for consecutive blocks of the sample of random variable D. When a block of key D'_i is demanded, one compute $D'_i = \mathcal{H}(i, D_{\sigma(i,1)}, \ldots, D_{\sigma(i,d)})$ for $D_{\sigma(i,j)}$ being left nodes and σ a function describing graph $\mathcal{G}_\sigma$. We assume that disperser graph $\mathcal{G}_\sigma$ is publicly known.

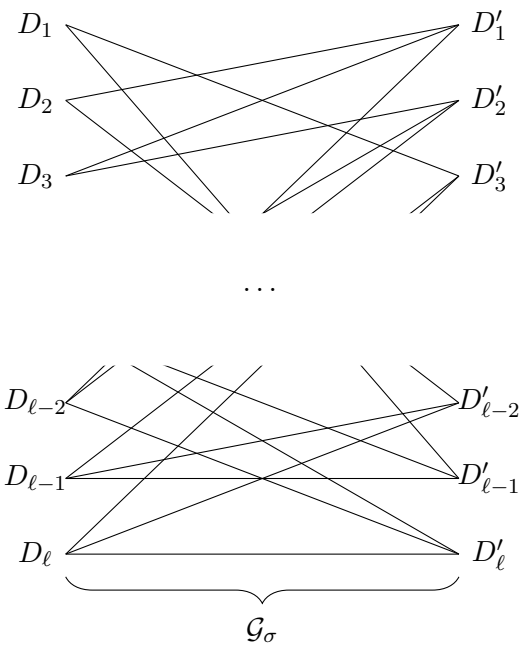


Figure 1.3.: An exemplary $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D)$ built on a 3-regular right disperser $\mathcal{G}_\sigma$.

One-wayness One of the main properties of our construction that implies both security and privacy is one-wayness of Disperse. Informally, we define it by saying that if adversary $\mathcal{A}$

- has access to all nodes D'_i on the right side of graph $\mathcal{G}_\sigma$,
- has no access to left nodes $D_{\sigma(i,j)}$,
- is given access to a leakage oracle $\mathcal{O}_\lambda^{D,\mathcal{H}}$ limited by the leakage parameter λ,
- can submit up to $q(n)$ queries to a random oracle $\mathcal{H}$.

then the number of queries to $\mathcal{H}$ equal $\left(i, D_{\sigma(i,1)}, \ldots, D_{\sigma(i,d)}\right)$ is strictly limited beside negligible probability.

This property is crucial in the presented work, we believe it could be useful anywhere where is a demand for a function that no one can obtain any useful information on the input from the function's output even under leakage. This property could be described as a stronger notion of one-wayness known from one-way function [KL07].

1.4.5. Identification and signature scheme on kdf

Except a concrete instantiation of a kdf function, we provide examples of identification and signature schemes that fit kdfs well. Both are based on a structure called Merkle tree [Mer79], which is often used in identification schemes because of its simplicity and very short public key. The signature scheme is delivered directly from the identification protocol by the Fiat-Shamir heuristic [FS87, FFS88].

1.4.6. Key refreshing

As mentioned before, one motivation for this work was to make a practical BRM scheme that can be run on devices with limited resources. Here, we derive the key from data already stored on the device. This approach comes with a major drawback. The key changes every time when the underlying data is modified. Since allowing user to modify data is a must, an efficient protocol that updates the key when data change is necessary too. The thesis delivers a solution for this problem that is simple and has communicational and computational complexity of a single identification.

1.5. Trivial solutions

In this part we briefly describe some naïve solution to the problem stated above.

Extractors We start with solution based on extractors [TV00, Tre01, Rao09]. We say that random variable X has average min-entropy greater than κ conditioned on variable L and write $\widetilde{\mathrm{H}}_\infty(X \mid L) > \kappa$ if $\kappa < -\log\left(\mathbb{E}_{l\in\mathrm{supp}(L)} \max_x \Pr(X = x \mid L = l)\right)$ (see Section 2.5 for more information about various notions of entropy). A (κ, ε)-extractor is a function such that for some uniformly random R over $\{0,1\}^d$, and input $X \in \{0,1\}^n$ outputs $\mathrm{Ext}(X, R) \in \{0,1\}^m$, a random variable such that $(\mathrm{Ext}(X, R), R, L)$ is ε-close (in terms of statistical distance) to (U_d, R, L) if $\widetilde{\mathrm{H}}_\infty(X \mid L) > \kappa$, U_d denote a random variable of uniform distribution over $\{0,1\}^d$ and L is some additional information on X. Since the output of an extractor is close to the uniform and use of such a function does not require any additional assumptions it seems a good idea to use them instead of more complex construction of dispersers and random oracle. Unfortunately, it is not. Here we mention two important arguments against such a solution:

- In our solution we do not assume that there are some data on the device hidden to the adversary, she has an access to everything the legitimate user has. Thus, extractor randomness R is available to $\mathcal{A}$ as well and it cannot be prevented that the adversary granted with access to a leakage oracle gets some information on R and perform leakage operations on X depending of obtained information. This makes X and R (possibly) dependent. Unfortunately, aforementioned extractor definition does not consider such a possibility.

- Due to the size of stored data, BRM schemes make use only of some constant (in terms of key size) number of bits of key. Thus fast computation of a part of key is a must. Unfortunately, solution based on extractors usually do not provide such a feature and outputs the whole key even if only a small part of it is demanded. This makes solution based on extractors impractical, because of big computation overhead.

Block by block hashing Another potential solution is to process data block by block. Consider a random oracle $\mathcal{H}$ and consecutive blocks of data $D_1, \ldots, D_\ell$. The obtained key equals $D'_1 = \mathcal{H}(1, D_1), \ldots, D'_\ell = \mathcal{H}(\ell, D_\ell)$. Our solution proposed in Chapter 3 assumes only some amount of min-entropy in D and it does not demand equal distribution of min-entropy among blocks $D_1, \ldots, D_\ell$. Hence, a situation where some blocks are determined, e.g. they correspond to some fixed part of files like headers, is allowed, what gives an additional information for the adversary. Furthermore, some blocks may equal each other, what allows the adversary to get information about two parts of the output key covering the cost of leaking a value of a single block only. Thus, it seems reasonable to deliver a solution where every part of the output depends on many blocks of input data to make sure that dependencies between the key blocks are not straight-forward. This is indeed the case in the proposed solution.

1.6. Organization of the thesis

Chapter 2 is dedicated to preliminaries on security games, Random Oracle Model and leakage-resilient cryptography. Furthermore, we introduce various notions of entropy and point out important properties of conditional min-entropy. We also discuss the difference between the real world and the ideal world and recall the notion of simulator. Next, we recall some basic cryptographic primitives like signature and identification schemes. We define these notions for a leakage-resilient setting. Last but not least, we show basic properties of disperser graphs that are an important building block in this thesis.

Chapter 3, contains the main result of this thesis and presents a key derivation function. Even though kdfs have been proposed in [DY13], we present there a new definition for this class of functions. This approach is justified by introducing a new requirement on kdfs, that is, privacy. The chapter also presents an exemplary kdf function called Disperse. We show that obtained function fulfils our requirements, i.e. is both private and secure. In this chapter we also define a game Guessing and shows one-wayness of Disperse function that may be interesting of their own.

Chapter 4 provides potential use cases for Disperse function and shows examples of identification and signature schemes. Both schemes are based on a well-known primitive called Merkle tree. However, we are probably the first who shown that these constructions are secure in the Bounded Retrieval Model. In this chapter we also propose a method to overcome the problem of modifying data used to produce the key.

Chapter 5 states some open problems due to the further work. Appendix A contains proofs omitted in the main part of the thesis.

CHAPTER 2

PRELIMINARIES

The following chapter contains a short description of a computational model and presents cryptographic primitives which are used in this thesis. It also introduces some basic cryptographic notions and provides short but essential information on min-entropy as a security measure. Furthermore, it provides elementary facts on bipartite disperser graphs, which properties are extensively used in this thesis.

2.1. Security games

Security of cryptography schemes are usually proven by defining a *game*, called from now on a *security game*, played between two interactive algorithms – one called an adversary, denoted by $\mathcal{A}$, and another called a challenger, denoted by $\mathcal{C}$. The adversary is an algorithm that tries to break the security of a scheme, while the challenger is a counterpart that interacts with her. We point out that such an interaction is crucial since it models that the adversary can learn something from observing scheme executions. We say that the adversary wins the game if she makes the challenger to output `Accept`. Otherwise, $\mathcal{A}$ loses and $\mathcal{C}$ outputs `Reject`. We say that the adversary breaks the security of the game if she wins the corresponding security game with

- non-negligible probability, or
- probability non-negligibly greater than $1/2$.

The case depends on the game played. For the former see the game on Fig. 2.2. For the latter, see Ex. 1. Below we formalize the notion of a security game.

Definition 1 (Security game[1])**.** A *security game* with randomness R against interactive algorithm $\mathcal{A}$ is a tuple

$$\mathsf{Game} = (\mathcal{C}, \mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Setup}_{\mathcal{C}}, \mathsf{Setup}_{\mathcal{A}}, \mathsf{Execute})$$

consisting of an interactive algorithm $\mathcal{C}$, called the challenger, together with a randomized public parameters generation procedure $\mathsf{ParamGen}$, key generation function KeyGen, a pair of setup procedures $\mathsf{Setup}_{\mathcal{C}}$, $\mathsf{Setup}_{\mathcal{A}}$ and an execution procedure $\mathsf{Execute}$ which given an interactive algorithm $\mathcal{A}$ operates as described in Figure 2.1.

Execution procedure Execute **of**
$\mathsf{Game} = (\mathcal{C}, \mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Setup}_{\mathcal{C}}, \mathsf{Setup}_{\mathcal{A}}, \mathsf{Execute})$**.**

1. System parameters are generated $\mathtt{params} \leftarrow \mathsf{ParamGen}(1^n)$, for security parameter n.
2. The key is initialized by $K \leftarrow \mathsf{KeyGen}(1^n; R)$, for randomness R. Then the input tapes of $\mathcal{A}$ and $\mathcal{C}$ are set to $\mathsf{Setup}_{\mathcal{A}}(K, \mathtt{params})$ and $\mathsf{Setup}_{\mathcal{C}}(K, \mathtt{params})$ respectively.

Execution phase:

3. The following loop is conducted:

Algorithm 1: Main loop

```
1 msg_{C↦A} = ⊥ /* first message is empty                    */
2 while state_C ∉ {Accept, Reject} do
3 |   A : (state_A, msg_{C↦A}) ⇝ (state'_A, msg_{A↦C})
4 |   C : (state_C, msg_{A↦C}) ⇝ (state'_C, msg_{C↦A})
5 end
```

Output: The final state $\mathsf{state}_{\mathcal{C}} \in \{\mathtt{Accept}, \mathtt{Reject}\}$ of the challenger $\mathcal{C}$.

Figure 2.1.: Execution procedure

For an interactive machines $\mathcal{X}, \mathcal{Y}$, such that $\mathcal{X}, \mathcal{Y} \in \{\mathcal{A}, \mathcal{C}\}$ and $\mathcal{X} \neq \mathcal{Y}$, we use notation $\mathcal{X} : (\mathtt{state}_{\mathcal{X}}, \mathtt{msg}_{\mathcal{Y}\mapsto\mathcal{X}}) \rightsquigarrow (\mathtt{state}'_{\mathcal{X}}, \mathtt{msg}'_{\mathcal{X}\mapsto\mathcal{Y}})$, which indicates that the interactive machines $\mathcal{X}$ resumes in state $\mathtt{state}_{\mathcal{X}}$ given an input message $\mathtt{msg}_{\mathcal{Y}\mapsto\mathcal{X}}$ from a machine $\mathcal{Y}$ and transits to state $\mathtt{state}'$ with an output message $\mathtt{msg}'_{\mathcal{X}\mapsto\mathcal{Y}}$ to $\mathcal{Y}$. Given all the parameters, we denote the execution of the game by $\mathsf{Game}[\mathcal{A} \leftrightharpoons \mathcal{C}, K \leftarrow \mathsf{KeyGen}(1^n; R)]$, where $\leftrightharpoons$ stands for a communication between parties $\mathcal{A}$ and $\mathcal{C}$. Note, we will usually

[1]The definition in the presented form originated in [DDK+16]. However, it may be seen as a version of the definition from [GW11].

skip the second argument of KeyGen, i.e. the randomness if it is taken from the uniform distribution. Note also, that in the latter part of the thesis we assume $\mathsf{KeyGen}(1^n, R) = R$, see discussion in Section 1.4.3.

Intuitively, the operation of $\mathsf{Game}[\mathcal{A} \leftrightharpoons \mathcal{C}, K \leftarrow \mathsf{KeyGen}(1^n; R)]$ boils down to an adaptive, sequential (numbered by round) exchange of messages between interactive machines $\mathcal{A}$ and $\mathcal{C}$ initialized by the values $\mathsf{Setup}_{\mathcal{A}}(K, \mathtt{params})$ and $\mathsf{Setup}_{\mathcal{C}}(K, \mathtt{params})$ respectively, which ends up in the last state of the algorithm $\mathcal{C}$. Having defined what a security game is, we define what does it mean that adversary $\mathcal{A}$ wins it and a game security notion.

Definition 2 (Game security[2])**.** Let **TM** be some collection of Turing machines. We say that $\mathsf{Game} \leftarrow (\mathcal{C}, \mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Setup}_{\mathcal{C}}, \mathsf{Setup}_{\mathcal{A}}, \mathsf{Execute})$ based on randomness K is $(\varepsilon(n), \mathbf{TM})$-secure, for some security parameter n, iff for every $\mathcal{A} \in \mathbf{TM}$ the probability that execution of $\mathcal{C}$ ends in $\mathtt{Accept}$ satisfies

$$\Pr(\mathsf{Game}[\mathcal{A} \leftrightharpoons \mathcal{C}, K \leftarrow \mathsf{KeyGen}(1^n; R)] = \mathtt{Accept}) \leqslant \varepsilon(n) \ , \tag{2.1}$$

where the probability is taken over R and all random choices of $\mathcal{C}$ and $\mathcal{A}$.

We usually require that ε given above is a negligible function or $\varepsilon(n) = \frac{1}{2} + \varepsilon'(n)$ for a negligible ε'.

Definition 3 (Negligible function, see [KL07])**.** We call a function $\varepsilon : \mathbb{N} \to \mathbb{R}$ negligible, if for every polynomial function *poly* there exists $N \in \mathbb{N}$ such that for every $n > N$ holds $\varepsilon(n) < \frac{1}{poly(n)}$.

Recall Ex. 1 for an example of a security game. This particular game is an indistinguishability game, i.e. we say that the adversary *wins* the game if she is able to distinguish between two worlds with probability non-negligibly better than 1/2. On the other hand we have another example, see Figure 2.4, of a game where the adversary *wins* the game if she come up with a valid signature on a message with non-negligible probability. The definition above covers both cases.

Remark. Recall that for every secret key k from Ex. 1 there is a machine $\mathcal{A}$ such that $\mathcal{A}$ wins the game always. E.g. if k is a part of $\mathcal{A}$'s code, then $\mathcal{A}$ can simply decrypt message $\mathsf{Enc}_k(m_b)$ and tell whether it contains encryption of m_0 or m_1. However, probability that such "well-fitted" k is picked is $1/|\mathbf{KeySpace}|$, which is negligible. Thus, Eq. 2.1 should be interpret as follows: pick adversary $\mathcal{A}$ and set public parameters $\mathtt{params}$. Finally, run KeyGen algorithm and obtain secret key K (uniformly random over predefined **KeySpace**).

Usually we require that the challenger is a probabilistic polynomial time (PPT) Turing machine:

[2]The definition in the presented form originated in [DDK+16]. However, it may be seen as a version of the definition from [GW11].

Definition 4 (Probabilistic polynomial time Turing machine (PPT), see [KL07])**.** We say that Turing machine $\mathcal{A}$ is PPT iff $\mathcal{A}$ have access to an auxiliary tape with random input R and on input x machine $\mathcal{A}$ outputs $\mathcal{A}(x;R)$ in time polynomial in $|x|$.

2.2. Real world vs ideal world

One of the methods for proving security of a cryptographic primitive is to show that an adversary who observes primitive's behaviour cannot learn anything important (from the security point of view) on its internals and inputs. This idea has been formalized by introducing concepts of ideal and real world. In the ideal world we assume that the adversary (by definition) learns nothing valuable from an interaction with the primitive. On the other hand, we have the real world, were we do not make such an assumption and have to take into consideration that the adversary may be able to learn something. If both worlds are indistinguishable, we conclude that our real-world primitive is secure – since the distinguisher cannot distinguish the ideal world where she learns nothing from the other, the real world, it means she learns nothing in the real world as well.

Of course, distinguishing algorithm can always toss a coin and guess randomly. Thus, we demand that no algorithm can distinguish with probability non-negligibly better than $\frac{1}{2}$.

For example consider semantic security of an encryption scheme (see, e.g. [Gol04, GM84]). In the ideal world the adversary that submits plaintext learns corresponding ciphertext and nothing more. On the other hand, in the real world situation may be a little different. Suppose that the adversary may be able to come up with a probability ensemble $(X_n)_{n\in\mathbb{N}}$ of plaintexts that allows her to learn some information about a plaintext from the corresponding ciphertext. In such case the distinguisher, which also have this information, is able to distinguish the worlds, because in the ideal world she learns nothing by the definition. To show that distinguishing the worlds is not possible we introduce a machine, called henceforth a simulator and denoted by $\mathcal{S}$, that given access to the primitive in the ideal world *simulates* everything the distinguisher may see.

Since both the adversary and the simulator are Turing machines, we can equip the latter with adversary's code (as an additional tape). However, we do not equip the simulator with auxiliary tapes given to the adversary, like tape with adversary's randomness or adversary's input. In case of an encryption scheme, the simulator has an oracle access to the encryption scheme, can query it with plaintexts and get corresponding ciphertexts. Simulator does not have access to any internal secrets of the scheme, like a secret key, and it (usually) does not have any special power over the adversary. Since it does not know the adversary's queries or responses from the oracle, (since the simulator does not know $\mathcal{A}$'s randomness and input) the simulator provides encryption of some *garbage*, like encryption of $0^{|X_n|}$, instead of encryption of X_n. If the simulator succeeds and the distinguisher does not distinguish encryption of X_n from encryption of $0^{|X_n|}$ we can conclude that the adversary learns nothing from encryption of $(X_n)_{n\in\mathbb{N}}$.

We mentioned that no machine can distinguish the ideal and the real world and the best she can do is to guess which world she interacts with. That means we allow the distinguisher to guess with a slightly better probability over $\frac{1}{2}$ only if her advantage over random guess is negligible.

Below we define what does it mean that two probability ensembles are computationally indistinguishable.

Definition 5 (Computational indistinguishability, see [Lin17])**.** Let $\mathbf{X} = \{X(a,n)\}_{a\in\{0,1\}^*,n\in\mathbb{N}}$ and $\mathbf{Y} = \{Y(a,n)\}_{a\in\{0,1\}^*,n\in\mathbb{N}}$ be probability ensembles indexed by two variables $a \in \{0,1\}^*$ and $n \in \mathbb{N}$. We say that $\mathbf{X}$ and $\mathbf{Y}$ are computationally indistinguishable if for every non-uniform probabilistic polynomial time algorithm $\mathcal{D}$ there exists negligible function ε such that for every $a \in \{0,1\}^*$ and every $n \in \mathbb{N}$:

$$|\Pr(\mathcal{D}(X(a,n)) = 1) - \Pr(\mathcal{D}(Y(a,n)) = 1)| \leqslant \varepsilon(n) \ .$$

Regarding secure computation, index a can be interpreted as system parameters and n as a security parameter. We allow a to be different for every n. Regarding aforementioned example of encryption scheme security we can illustrate this definition by stating that distribution $\mathbf{X}$ has been delivered by adversary $\mathcal{A}$ communicating with a real world scheme, while the other distribution $\mathbf{Y}$ has been produced by simulator $\mathcal{S}$ equipped with access to the functionality in the ideal world.

We also introduce a stronger notion of indistinguishability here – statistical indistinguishability. It allows the adversary to be computationally unbounded (in fact, computational power of the adversary is not important here at all).

Definition 6 (Statistical indistinguishability, see [Lin17])**.** Let $\mathbf{X} = \{X(a,n)\}_{a\in\{0,1\}^*,n\in\mathbb{N}}$ and $\mathbf{Y} = \{Y(a,n)\}_{a\in\{0,1\}^*,n\in\mathbb{N}}$ be probability ensembles indexed by two variables $a \in \{0,1\}^*$ and $n \in \mathbb{N}$. We say that $\mathbf{X}$ and $\mathbf{Y}$ are statistically indistinguishable if for some negligible function ε and every $a \in \{0,1\}^*$ and every $n \in \mathbb{N}$:

$$\Delta(X(a,n), Y(a,n)) \leqslant \varepsilon(n) \ .$$

Simulation in leakage-resilient cryptography In case of leakage, we give the simulator additional power of making leakage queries as the adversary does. However, it may be difficult for the simulator to do its job being as restricted as the adversary is. Thus, we allow the simulator to have a slightly bigger leakage $\Lambda(\lambda)$, for a leakage parameter λ. We denote simulator's overhead $(\Lambda(\lambda) - \lambda)$ by Δ_λ. This gap between simulator's and adversary's power makes our proofs less tight. That is, equipped with the simulator allowed to make leakage of size $\Lambda(\lambda)$ we can only proof security of a scheme against adversaries allowed to make leakage of size at most λ. Hence, we care for Δ_λ to be as small as possible.

2.3. Random Oracle Model

Many years of cryptographic research have shown that a vast number of proofs (and henceforth cryptographic schemes) would become much (conceptually) simpler if we were equipped with oracle access to a machine that takes some arbitrary, yet well defined, input and gives some random, unpredictable output. Furthermore, some proofs and constructions were shown impossible to achieve if no such oracle exists. We require that the machine behaves as a function – queried twice with the same input, outputs the same result. Existence of such machine has been formalized as the Random Oracle Model (ROM).

Random Oracle Model, introduced in [BR93], allows every algorithm in the protocol to have access to an interactive Turing machine $\Omega_{\mathbf{X},\mathbf{Y}}$ (called henceforth *random oracle*) containing a function $\mathcal{H} : \mathbf{X} \to \mathbf{Y}$ picked uniformly at random from a class of all functions mapping $\mathbf{X}$ to $\mathbf{Y}$, where $\mathbf{X}, \mathbf{Y}$ are finite sets, i.e. it is a random function. Every algorithm can query machine $\Omega_{\mathbf{X},\mathbf{Y}}$ on input $x \in \mathbf{X}$ and obtain $\mathcal{H}(x) \in \mathbf{Y}$. (For simplicity, we will usually say that machine queries function $\mathcal{H}$ itself.) Access to Ω is black-box. More precisely, a machine with access to Ω can send queries x and learn the corresponding output $\mathcal{H}(x)$. However, it cannot observe how the oracle works.

In this model we (sometimes) do not limit adversary's computational power or available memory space. Instead we assume that the number of queries the adversary can submit to the oracle is bounded by some function $q(n)$. This thesis has been done in such setting.

Impossibility of a random oracle It is necessary to emphasize that no deterministic function can be a random oracle. This has been proven by Canetti et al. in [CGH04] and independently by Nielsen in [Nie02] by showing that a random oracle yields properties impossible to achieve in the standard model. However, using a random oracle is well motivated by commonly substituting it by a hash function (like SHA-3, SHA-2 nowadays or MD5 in the past) in software and, more recently, hardware implementations.

There is an ongoing discussion on using a random oracle in cryptographic proofs. On the one hand, the standard model is assumed to fit the real world better. Furthermore, this model needs less assumptions and usually less assumptions is better.

On the other hand, there are rational arguments on the opposite point of view. As pointed by Menezes and Koblitz in [KM15], proofs that a random oracle cannot be securely instantiated by a hash function uses *unnatural* constructions with no practical application whatsoever. Furthermore, the authors state that replacing schemes secure in ROM by much more complicated schemes in the standard model leads to inevitable more tangled implementation, which is more vulnerable to implementation-based attacks and less efficient. Another argument provided by the authors points that the random oracle assumption is often replaced by another assumption, less known and understood, making the security model even less realistic.

2.4. Leakage-resilient cryptography

The way we use cryptographic schemes in the real world sometimes does not fit to an idealized usability predicted by the scheme inventors. A number of real-world attacks on cryptographic primitives were possible due to their faulty implementation or additional knowledge an adversary obtained by measuring physical features (like electro-magnetic radiation). These attacks have shown that an idealized model that assumes an adversary, which has no knowledge whatsoever of secrets used in a scheme, is not enough and more sophisticated model is necessary.

In leakage-resilient cryptography we allow adversary $\mathcal{A}$ to learn some *leaked* information on secret X. This knowledge is formalized by equipping $\mathcal{A}$ with black-box access to a leakage oracle $\mathcal{O}_\lambda^X$. Oracle $\mathcal{O}_\lambda^X$ contains X as an additional parameter. This parameter is not known by $\mathcal{A}$. Queries to $\mathcal{O}$ made by the adversary are descriptions of functions $\{f_i\}_{i\in[N]}$, such that $f_i : \{0,1\}^{|X|} \to \{0,1\}^{\lambda_i}$, for some $N \in \mathbb{N}$ and $\lambda_i \in \mathbb{N}$. The size of the description of f_i is irrelevant here (it can be even exponential in $|X|$). Obviously this only makes our model stronger. The oracle answers on adversary's query f_i with $f_i(X)$ if, and only if $\sum_{j\leq i} \lambda_j \leqslant \lambda$. We call λ a *leakage parameter* or a *leakage bound*. We will usually write just $\mathcal{O}^X$ instead of $\mathcal{O}_\lambda^X$, when λ is clear from the context.

We denote by $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}^X,\mathcal{H}}_{\lambda,q(n)}$ a Turing machine $\mathcal{A}$ granted with an oracle access to

- a leakage oracle $\mathcal{O}^X$ with leakage bounded by λ and
- up to $q(n)$ queries to a random oracle $\mathcal{H}$, for the security parameter n.

Relative leakage We say that a protocol is secure in the relative leakage model if the leakage parameter λ of adversary $\mathcal{A}$ is bounded by some function $p : \mathbb{N} \setminus \{0\} \to \mathbb{N}$ with property $p(x) < x$. Leakage parameter λ depends on the length of a secret key. This model has a straightforward motivation. It covers well attacks based on imperfect reading of devices' memory, like the cold-boot attacks described in [HSH+08]. Note that in the relative leakage model there is no need for large secret keys. Assume a scheme secure against leakage λ such that $\lambda \leqslant |\mathtt{sk}|$. The scheme is secure in a case where `sk` is a few kilobytes long, as well as in a case when `sk` is really large.

Absolute leakage, Bounded Retrieval Model We assume that the upper bound of leakage λ is given *from the outside*, as a system parameter. It is absolute and independent from the key length. The bound is given once for the lifetime of the system. Contrary to the relative leakage, we set here a length of a key according to λ. Thus, a huge leakage parameter yields a huge secret key.

However, a huge secret key usually implies computational inefficiency of a scheme, hence in the BRM scheme's computational and communication complexity depends at most polylogarithmically on λ. In this thesis we employed this model of leakage.

2.5. Various notions of entropy

Suppose that adversary $\mathcal{A}$ is a probabilistic polynomial time machine. Security proof against such an adversary is usually based on a reduction from a well-known security assumption, called *hardness assumption*, to breaking the scheme. A hardness assumption is a computational or a decisional problem that is considered hard, i.e. it has not known solution that works in polynomial time.[3] For example RSA assumption, factorization, discrete logarithm problem, just to name a few. By presenting the reduction, we prove that the assumption is at most as hard to break as the presented scheme.

On the other hand, we have information-theoretic model of security that does not rely on hardness assumptions. In this model no computational limitations of adversary's power have place. Instead, proofs use sophisticated methods taken directly from the probability theory. While computational security shows that some function (like extracting secret key from a public one) is hard to compute, information theoretic proofs show that despite of computational effort, the output of a desired function preserves its *uncertainty*. The measures for aforementioned *uncertainty* are different notions of entropy, which are presented below.

Shannon entropy In his seminal work [Sha48] Shannon introduced a definition of entropy of a random variable, a measure determining variable's uncertainty. That put the ground for a whole new field of computer science, i.e. information theory.

Remark. Every logarithm in this work has base 2.

Definition 7 (Shannon entropy, see [CT91])**.** Shannon entropy of a random variable X, denoted by $\mathrm{H}(X)$ is defined by

$$\mathrm{H}(X) = -\sum_{x} \Pr(X = x) \log \Pr(X = x) \ ,$$

where $x \in \mathrm{supp}(X)$. We assume that $\log \Pr(X = x) = 0$ for $\Pr(X = x) = 0$.

This definition expands nicely to joint and conditional distributions of random variables:

Definition 8 (Joint entropy, see [CT91])**.** The joint entropy of random variables $X_1, \ldots, X_n$ is expressed by:

$$\mathrm{H}(X_1, \ldots, X_n) = \sum_{x_1} \ldots \sum_{x_n} \Pr(X_1 = x_1, \ldots, X_n = x_n) \log(\Pr(X_1 = x_1, \ldots, X_n = x_n)) \ ,$$

where $x_1 \in \mathrm{supp}(X_1), \ldots, x_n \in \mathrm{supp}(X_n)$.

[3] Note that usually we cannot provide a proof that a cryptographic problem is NP-hard. Furthermore in cryptography we require problems that are hard on average not in the worst case only.

Definition 9 (Conditional entropy, see [CT91])**.** The conditional entropy of random variables X, Y equals

$$\begin{aligned}\mathrm{H}(X \mid Y) = &\sum_{y \in \mathrm{supp}(Y)} \Pr(Y = y)\, \mathrm{H}(X \mid Y = y) = \\ &- \sum_{y \in \mathrm{supp}(Y)} \sum_{x \in \mathrm{supp}(X)} \Pr(X = x, Y = y) \log \Pr(X = x \mid Y = y) \ .\end{aligned}$$

From the aforementioned definitions we conclude the following useful property of entropy which expresses joint entropy of two random variables in terms of entropy and conditional entropy:

Theorem 1 (Chain rule, see [CT91])**.**

$$\mathrm{H}(X, Y) = \mathrm{H}(X) + \mathrm{H}(Y \mid X) \ .$$

Proof. The proof can be found in [CT91]. □

Min-entropy Despite of significant role Shannon entropy plays in computer science, this particular notion of uncertainty is not very useful in cryptography. An illustrative example below shows how delusive may be relying on it:

> **Example 9.** Consider an encryption scheme, which takes a message m from a message space **MsgSpace**, applies a block-cipher function f with key distribution K over **KeySpace** $= \{0,1\}^n$. We assume that such a scheme is secure for K of uniformly random distribution, but what if K is not uniformly distributed? For instance let $\Pr(K = 1^n) = \frac{1}{2}$ and $\Pr(K = k \mid k \neq 1^n) = \frac{1}{2(2^n - 1)}$. Obviously, if we take the key from such distribution, we cannot claim that our encryption is secure. The adversary may just guess the key with high probability. However, according to Shannon's definition of entropy,
>
> $$\begin{aligned}\mathrm{H}(K) &= -\sum_{k \in \mathrm{supp}(K)} \Pr(K = k) \log \Pr(K = k) = \\ &= -\left(\frac{1}{2} \cdot (-1) + (2^n - 1)\, \frac{1}{2(2^n - 1)} \cdot \left(\log \frac{1}{2(2^n - 1)}\right)\right) \geqslant \\ &\geqslant \frac{1}{2} - \frac{1}{2} \cdot \left(\log \frac{1}{2^{n+1}}\right) = \frac{1}{2} - \frac{1}{2} \cdot (-n - 1) = \frac{1}{2} + \frac{(n+1)}{2} \ .\end{aligned}$$

Despite the fact that K has cryptographically useless distribution, it has still high entropy. Thus, another notion of randomness is necessary.

Definition 10 (Min-entropy, see [DORS08])**.** Min-entropy of random variable X, denoted

by $\mathrm{H}_\infty(X)$, is defined by

$$\mathrm{H}_\infty(X) = \min_{x \in \mathrm{supp}(X)} (-\log \Pr(X = x)) = -\log(\max_{x \in \mathrm{supp}(X)} \Pr(X = x)) \ .$$

Simple yet illustrative explanation of the idea of min-entropy is to tangle the measure of uncertainty of a random variable with the probability of its most probable output. Since the adversary can tune her strategy to win in the most probable case, this notion of randomness fits well to cryptographic purposes.

Recall the example with an useless distribution K of high (Shannon) entropy. We compute its min-entropy now.

$$\mathrm{H}_\infty(K) = -\log \max_{k \in \mathrm{supp}(K)} \Pr(K = k) = -\log \frac{1}{2} = 1 \ .$$

Note, the min-entropy is tiny and the gap between entropy and min-entropy is noticeable.

Conditional min-entropy As stated before, in any model where the adversary is granted leakage, she can obtain some additional information about the secret. In the last part we argued that min-entropy is a proper measure to check whether a random variable has some uncertainty or not. Thus, it seems a natural idea to introduce a notion of conditional min-entropy, which can give us an insight how uncertainty of a random variable behaves if someone obtained a partial information on it. This notion has been introduced by Dodis et al. [DORS08] and is defined below.

Definition 11 (Conditional min-entropy, [DORS08])**.** Conditional min-entropy of a random variable X given Y, denoted by $\widetilde{\mathrm{H}}_\infty(X \mid Y)$ is defined as:

$$\widetilde{\mathrm{H}}_\infty(X \mid Y) = -\log \left(\mathbb{E}_{y \in \mathrm{supp}(Y)} 2^{-\mathrm{H}_\infty(X|Y=y)} \right) \ . \tag{2.2}$$

This definition turns out to preserve the natural interpretation of min-entropy as the opposite to the logarithm of maximal probability of success in guessing X given Y. That is, given value y of a random variable Y the probability that the value of X is guessed is at most

$$\max_{x \in \mathrm{supp}(X)} \Pr(X = x \mid Y = y).$$

Thus, on average (depending of Y) probability that X is guessed is at most

$$\mathbb{E}_{y \in \mathrm{supp}(Y)} \left(\max_{x \in \mathrm{supp}(X)} \Pr(X = x \mid Y = y) \right) = \mathbb{E}_{y \in \mathrm{supp}(Y)} \left(2^{-\mathrm{H}_\infty(X|Y=y)} \right) = 2^{-\widetilde{\mathrm{H}}_\infty(X|Y)} \ . \tag{2.3}$$

We found very useful to have inequalities describing dependencies between min-entropy and conditional min-entropy. Especially, the following inequalities implying lower bounds for min-entropy given conditional min-entropy have been employed in the following parts

of this thesis.

Lemma 2 (Lemma 2.2 in Dodis et al. [DORS08])**.** *Let X, Y, Z be random variables. Then*

a) For any $\delta > 0$, the conditional entropy $\mathrm{H}_\infty(X \mid Y = y)$ is at least $\widetilde{\mathrm{H}}_\infty(X \mid Y) - \log(1/\delta)$ with probability at least $1 - \delta$ over the choice of y.

b) If Y has at most 2^λ possible values, then $\widetilde{\mathrm{H}}_\infty(X \mid (Y, Z)) \geqslant \widetilde{\mathrm{H}}_\infty((X, Y) \mid Z) - \lambda \geqslant \widetilde{\mathrm{H}}_\infty(X \mid Z) - \lambda$. In particular, $\widetilde{\mathrm{H}}_\infty(X \mid Y) \geqslant \mathrm{H}_\infty(X, Y) - \lambda \geqslant \mathrm{H}_\infty(X) - \lambda$.

2.6. Identification scheme

One of the most important cryptographic primitives are identification schemes. These schemes allow one party to identify to another remote party. Without identification schemes no internet banking, privacy of emails or any social media like Facebook are possible. An effective and secure identification scheme is necessary to any secure internet connection.

We define a 3-round identification scheme along with a security game for it. Such a scheme consists of two parties, a prover $\mathcal{P}$ and a verifier $\mathcal{V}$.

Definition 12 (Identification scheme, see [KL07])**.** Identification scheme $\Pi = (\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$ is an interactive protocol between two Turing machines, prover $\mathcal{P}$ and verifier $\mathcal{V}$, with the following subroutines:

Setup stage: $\mathtt{params} \leftarrow \mathsf{ParamGen}(1^n)$ for the security parameter n, $\mathtt{params}$ contains description of a message space **MsgSpace**, challenge space **ChlSpace** and response space **RspSpace**; the keys are generated as $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{KeyGen}(1^n; R)$ for randomness R; prover $\mathcal{P}$ gets a triple $(\mathtt{pk}, \mathtt{sk}, \mathtt{params})$ and verifier $\mathcal{V}$ gets a pair $(\mathtt{pk}, \mathtt{params})$.

Challenge: Prover $\mathcal{P}$ sends a message $a \in$ **MsgSpace**, which is followed by verifier's challenge $c \in$ **ChlSpace**. On this challenge, $\mathcal{P}$ replies with $z \leftarrow \mathsf{Prove}(\mathtt{sk}; a, c) \in$ **RspSpace**.

Verification: Verifier $\mathcal{V}$ runs $\mathsf{Verify}(\mathtt{pk}; a, c, z)$ and outputs `Accept` if $\mathsf{Verify}(\mathtt{pk}; a, c, z)$ does so and `Reject` in the other case. Intuitively, $\mathsf{Verify}(\mathtt{pk}; a, c, z)$ accepts iff z has been computed honestly, accordingly to message a, challenge c and the prover's secret key $\mathtt{sk}$.

Below we define the security of an identification scheme.

Definition 13 (Security of an identification scheme, see [ADW09])**.** We say that an identification scheme Π is $(\varepsilon(n), \mathbf{TM})$-secure if for every adversary $\mathcal{A}$, a Turing machine from **TM**, the probability of winning the game described at Figure 2.2 is at most $\varepsilon(n)$. The adversary wins if verifier $\mathcal{V}$ accepts the conversation taking place during the impersonation stage. We will omit the second parameter in the security definition if the class the

Identification scheme security game

Parameters: n – security parameter.

1. **Setup stage:** $\mathtt{params} \leftarrow \mathsf{ParamGen}(1^n)$; $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{KeyGen}(1^n; R)$; $\mathcal{A}$ gets $(\mathtt{pk}, \mathtt{params})$.
2. **Test stage:** In this stage, $\mathcal{A}^{\mathcal{P}(\mathtt{pk},\mathtt{sk})}$ has an oracle access to $\mathcal{P}(\mathtt{pk}, \mathtt{sk})$. The adversary submits to the oracle at most polynomially many queries.
3. **Impersonation stage:** Having lost access to $\mathcal{P}(\mathtt{pk}, \mathtt{sk})$, adversary $\mathcal{A}$ tries to identify herself as a legitimate prover $\mathcal{P}$ to verifier $\mathcal{V}$. The verifier outputs `Accept` if the adversary succeeds and `Reject` otherwise.

Figure 2.2.: ID_n - identification security game, cf. [ADW09].

adversary belongs to is clear from the context. Usually we pick **TM** to be a class of PPT machines. We say that an identification scheme is secure if it is $(\varepsilon(n), \mathbf{TM})$-secure for some negligible $\varepsilon(n)$ and **TM** clear from the context.

2.6.1. Making identification non-interactive, Fiat-Shamir paradigm

In the random oracle model, an interactive 3-round identification scheme can be easily made non-interactive by so-called Fiat-Shamir paradigm [FS87]. Suppose that identification protocol consists of three messages $(a, c, \mathsf{Prove}(\mathtt{sk}; a, c))$, where the first comes from the prover, the second comes from the verifier and the third is prover's response to verifier's challenge c that also depends on the first message a. An identification protocol that allows to publish verifier's random choices, here the challenge, is called *public-coin*.

Fiat-Shamir approach transforms any such public-coin 3-message argument system into a non-interactive argument by using a random oracle, see Figure 2.3. The main argument here is as follows. Assume that some set of challenges $\mathbf{C}$ fits a malicious prover $\mathcal{P}^*$, who does not know `sk`, well. That is, if a challenge comes from $\mathbf{C}$ then $\mathcal{P}^*$ can answer it. Since $\mathcal{H}(a)$ does not reveal any information on a, malicious prover has only negligible chances of coming up with such message a that $\mathcal{H}(a) \in \mathbf{C}$.

However, to use Fiat-Shamir paradigm we need to assume that the verifier picks her challenge uniformly at random. Assume the opposite, identification scheme verifier is not honest, deviates from the protocol and picks her challenges from some non-uniform distribution. Clearly, challenges from such malicious verifier cannot be substituted by oracle answers since they are taken from a uniform distribution.

2.7. Signature schemes

Another important primitive crucial to modern day communication are signature schemes. As the name suggests, signature schemes allow a party to *sign* a message. That is, provide

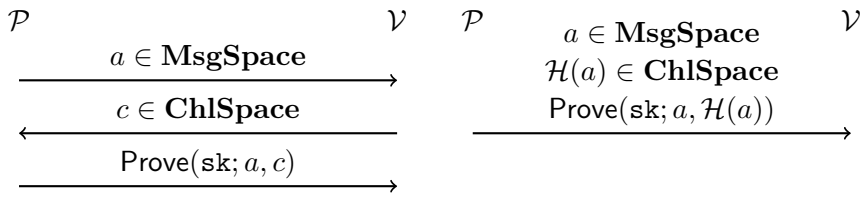


Figure 2.3.: Non-interactive Fiat-Shamir proof system compared to 3-message public-coin proof of argument.

a proof that the signer confirmed the content of the signed document. Importantly, this primitive assures the integrity of the message, since any change in the text implies a change in the output signature (with overwhelming probability). We call the party who signs a prover $\mathcal{P}$, and another which verifies signature, a verifier $\mathcal{V}$.

Definition 14 (Signature scheme, see [KL07])**.** Signature scheme $\Sigma = (\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is an interactive protocol between two Turing machines a prover $\mathcal{P}$ and a verifier $\mathcal{V}$ with the following subroutines:

Setup phase: $\mathtt{params} \leftarrow \mathsf{ParamGen}(1^n)$ defines system parameters and message space **MsgSpace** getting security parameter n as input. Algorithm $\mathsf{KeyGen}(1^n; R)$ outputs signature and verification keys $(\mathtt{sigk}, \mathtt{verk})$ for randomness R.

Signing: $\mathcal{P}$ runs $\mathsf{Sign}(\mathtt{sigk}; m)$, takes signature key $\mathtt{sigk}$ output by KeyGen, message $m \in \mathbf{MsgSpace}$ and returns signature σ.

Verification: $\mathcal{V}$ runs $\mathsf{Verify}(\mathtt{verk}; \sigma, m)$, takes verification key, message m, string σ and outputs $\mathtt{Accept}$ if σ is a correct signature on message m under key $\mathtt{sigk}$.

Security of a signature scheme is defined by the following game:

Definition 15 (Existential unforgeability, see [ADW09])**.** We say that a signature scheme $(\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is $(\varepsilon(n), \mathbf{TM})$-existentially unforgeable if every adversary $\mathcal{A} \in \mathbf{TM}$ makes $\mathcal{V}$ output $\mathtt{Accept}$ in the game described on Figure 2.4 with probability at most $\varepsilon(n)$. We say that a signature scheme is existentially unforgeable if it is $(\varepsilon(n), \mathbf{TM})$-existentially unforgeable for some negligible $\varepsilon(n)$. In such case, we assume that the class $\mathbf{TM}$ is clear from the context.

2.8. From identification schemes to signature schemes

Out of plethora of signature schemes, we provide an example that is derived directly from an identification scheme. Assume that identification scheme Π consists of three messages: statement a sent by the prover, challenge c provided by the verifier, followed by answer $z \leftarrow \mathsf{Prove}(\mathtt{sk}; a, c)$ for a secret key $\mathtt{sk}$. Assume that Π was made non-interactive by Fiat-Shamir paradigm, i.e. the verifier is honest and the challenge is $\mathcal{H}(a)$. Then the

Existential unforgeability of a signature scheme

Initialization System parameters and keys get set by $\texttt{params} \leftarrow \mathsf{ParamGen}(1^n)$ and $(\texttt{sigk}, \texttt{verk}) \leftarrow \mathsf{KeyGen}(1^n; R)$. Parameters $\texttt{params}$ are publicly known. Signing oracle $\mathcal{S}$ is parametrized by key $\texttt{sigk}$. $\mathcal{S}^{\texttt{sigk}}$ on input $m \in \mathbf{MsgSpace}$ outputs signature σ such that $\mathsf{Verify}(\texttt{verk}; \sigma, m)$ accepts. The adversary gets oracle access to $\mathcal{S}^{\texttt{sigk}}$. The verifier gets public parameters along with a verification key $\texttt{verk}$.

Signing queries Adversary $\mathcal{A}^{\mathcal{S}^{\texttt{sigk}}}$ performs up to polynomially many signing queries to oracle $\mathcal{S}^{\texttt{sigk}}$ thus learns up to polynomially many message-signature pairs.

Signing stage Adversary $\mathcal{A}^{\mathcal{S}^{\texttt{sigk}}}$ outputs a message, signature pair (m, σ) for a message m not queried before. Verifier $\mathcal{V}$ outputs $\texttt{Accept}$ if $\mathsf{Verify}(\texttt{verk}; \sigma, m)$ accepts.

Figure 2.4.: ExUG_n; unforgeability game for signature scheme, see [ADW09].

following signature scheme Σ is an existentially unforgeable signature scheme for any message $m \in \mathbf{MsgSpace}$ under the random oracle assumption:

Setup phase: $\Sigma.\mathsf{ParamGen}(1^n)$ gets $\texttt{params} \leftarrow \Pi.\mathsf{ParamGen}(1^n)$ and returns $\Pi.\mathsf{KeyGen}(1^n; R)$, i.e. $(\texttt{sigk}, \texttt{verk}) = (\texttt{sk}, \texttt{pk})$.

Signing: Prover $\mathcal{P}$ picks message m and some a and computes $\mathcal{H}(m, a)$, for a random oracle $\mathcal{H}$, and returns signature

$$\sigma \leftarrow \Sigma.\mathsf{Sign}(\texttt{sigk}; m) = ((m, a), \mathcal{H}(m, a), \Pi.\mathsf{Prove}(\texttt{sk}; (m, a), \mathcal{H}(m, a))) \ .$$

Verification: $\Sigma.\mathsf{Verify}$ takes (identification scheme) verification key $\texttt{pk}$ as verification key $\texttt{verk}$, string σ parsed as $((m, a), \mathcal{H}(m', a'), z)$, message m, and outputs $\Pi.\mathsf{Verify}(\texttt{pk}; (m, a), \mathcal{H}(m', a'), z)$.

The intuition behind the security reduction for this scheme is: if the adversary is able (with non-negligible probability) to break the security of the aforementioned scheme and sign a message m, i.e. produce valid triple $((m, a), \mathcal{H}(m, a), \Pi.\mathsf{Prove}(\texttt{sk}; (m, a), \mathcal{H}(m, a)))$ then she can also break the security of the identification scheme by setting (m, a) as a statement and responding to challenge $\mathcal{H}(m, a)$ with a proper $z \leftarrow \Pi.\mathsf{Prove}(\texttt{sk}; (m, a), \mathcal{H}(m, a))$.

2.9. Identification and signature schemes in the Bounded Retrieval Model

Because BRM allows the adversary to perform computation using users' data or secret keys (like $\texttt{sk}$ for identification and $\texttt{sigk}$ for signatures schemes), it is not possible to achieve

existential unforgeability. In a (well justified) case, where a signature is much shorter than the leakage, the adversary can simply leak a signature for a particular message of her choice. That was a motivation for Alwen et al. who introduced in [ADW09] so-called *entropic unforgeability*, a notion allowing to handle security properties of signatures exposed to a leakage, which has been described in Definition 18.

Definition 16 (Leakage-resilient identification scheme; see [ADW09])**.** We say that identification scheme $(\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$ is $(\varepsilon(n), \mathbf{TM}_\lambda^{\mathcal{O}^{\mathtt{sk}}})$-leakage-resilient with leakage parameter λ if any adversary $\mathcal{A} \in \mathbf{TM}_\lambda^{\mathcal{O}^{\mathtt{sk}}}$ has probability of winning $\mathsf{ID}_{\lambda,n}$ game (for security parameter n) described in Figure 2.5 not greater than $\varepsilon(n)$.

Definition 17 (Entropic adversary; from [ADW09])**.** For adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$, let $\mathsf{View}_{\mathcal{A}_1}$ be a random variable describing the view of $\mathcal{A}_1^{\mathcal{O}_\lambda^{\mathtt{sk}}, \mathcal{S}^{\mathtt{sigk}}}$ including her input, random coins, communication she observed and responses from both signing $\mathcal{S}^{\mathtt{sigk}}$ and leakage $\mathcal{O}_\lambda^{\mathtt{sk}}$ oracles.

Let $M_{\mathcal{A}_2}$ be a random variable describing the message output by $\mathcal{A}_2$ in $\mathsf{EUG}_{\lambda,n,\zeta(n)}$ (see Figure 2.8). We say that adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ is *n-entropic* if $\widetilde{\mathrm{H}}_\infty(M_{\mathcal{A}_2} \mid \mathsf{View}_{\mathcal{A}_1}) \geqslant n$, for security parameter n.

Remark. Given security parameter n we say that the adversary is *entropic* if she is n-entropic.

Definition 18 (Unforgeable sign. scheme under leakage; from [ADW09])**.**

1. We say that a signature scheme $(\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is $(\varepsilon(n), \mathbf{TM})$-*existentially-unforgeable* with leakage parameter λ if for any adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \in \mathbf{TM}$ in the game $\mathsf{EUG}_{\lambda,n}$ (see Figure 2.6 [4]) is at most $\varepsilon(n)$.
2. We say that the signature scheme is *entropically-unforgeable* with leakage parameter λ if the above holds only for all n-entropic adversaries $\mathcal{A}$.

Similarly to a non-leakage setting, a leakage-resilient identification scheme can be used to get a leakage-resilient signature. The construction is the same as presented in Section 2.8. The security of the new scheme has been proven by Alwen et al. [ADW09, Theorem 5.1] as below. The full proof of the theorem is given in Appendix A. We present the theorem with a proof (copied from the original paper, with small changes to make it consistent with the notation in this thesis), since we it will be used later in Corollary 4.

Theorem 3 (Theorem 5.1 from [ADW09])**.** *Let* $\Pi = (\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$ *by a public coins identification scheme consisting of three rounds of interaction initiated by the prover. Let* $\Sigma = (\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be the signature scheme produced by the Fiat-Shamir paradigm applied to* Π*. If* Π *allows leakage* λ*, then* Σ *is entropically-unforgeable with leakage* λ*.*

[4] This figure was presented accordingly to [ADW09].

Leakage-resilient identification scheme security game

Parameters: n – security parameter, λ – leakage parameter.

Setup stage: $\mathtt{params} \leftarrow \mathsf{ParamGen}(1^n)$; $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{KeyGen}(1^n; R)$; $\mathcal{A}$ gets $\mathtt{pk}, \mathtt{params}$

Test stage: In this stage, $\mathcal{A}^{\mathcal{O}_\lambda^{\mathtt{sk}}, \mathcal{P}(\mathtt{pk},\mathtt{sk})}$ has oracle access to $\mathcal{P}(\mathtt{pk}, \mathtt{sk})$ and submits there at most polynomially many proofs of identity. Furthermore, the adversary has access to the leakage oracle $\mathcal{O}_\lambda^{\mathtt{sk}}$.

Impersonation stage: Having lost an access to $\mathcal{P}(\mathtt{pk}, \mathtt{sk})$ and $\mathcal{O}_\lambda^{\mathtt{sk}}$, adversary $\mathcal{A}$ tries to identify herself as prover $\mathcal{P}$ to verifier $\mathcal{V}$. The verifier outputs `Accept` if the adversary succeeds and `Reject` otherwise.

Figure 2.5.: $\mathsf{ID}_{\lambda,n}$; leakage-resilient identification scheme security game, cf. [ADW09].

Leakage-resilient signature scheme entropic unforgeability security game

Parameters: n – security parameter, λ – leakage parameter.

Initialization System parameters and keys get set by $\mathtt{params} \leftarrow \mathsf{ParamGen}(1^n)$ and $(\mathtt{sigk}, \mathtt{verk}) \leftarrow \mathsf{KeyGen}(1^n; R)$. Parameters `params` are publicly known. There are signing oracle $\mathcal{S}$ parametrized by key `sigk` and leakage oracle $\mathcal{O}_\lambda^{\mathtt{sk}}$. The signing oracle on input $m \in$ **MsgSpace** outputs signature σ such that $\mathsf{Verify}(\mathtt{verk}; \sigma, m)$ accepts. The verifier gets public parameters along with the verification key `verk`.

Signing and leakage queries Adversary $\mathcal{A}_1^{\mathcal{O}_\lambda^{\mathtt{sk}}, \mathcal{S}^{\mathtt{sigk}}}$ is given access to the signing oracle $\mathcal{S}^{\mathtt{sigk}}$ and leakage oracle $\mathcal{O}_\lambda^{\mathtt{sk}}$. $\mathcal{A}_1$ outputs an arbitrary hint $v \in \{0,1\}^*$.

Post-leakage Entropic adversary $\mathcal{A}_2^{\mathcal{S}^{\mathtt{sigk}}}$ is given the hint v and access to the signing oracle $\mathcal{S}^{\mathtt{sigk}}$. The output of $\mathcal{A}_2$ is parsed as a message, signature pair (m, σ). Verifier $\mathcal{V}$ outputs `Accept` if $\mathsf{Verify}(\mathtt{verk}; \sigma, m)$ accepts.

Figure 2.6.: $\mathsf{EUG}_{\lambda,n}$; leakage-resilient signature scheme security game, cf. [ADW09].

2.10. Bounded number of executions

Our construction of an identification and a signature scheme in the BRM described later in Chapter 4 is based on a Merkle signature scheme [Mer79] which inherently works for a bounded number of executions between necessary key refreshing. We claim, that it is not a major problem in terms of usability, since for the real world parameters the upper bound for the number of executions $\zeta(n)$ can be as big as e.g. 100,000. That, for 10 identifications or signatures per day, gives a life-span of a particular key over 27 years. Considering security in such a long time, we can easily point out more important cryptographic problems than a must of a single key refreshing, especially if the refreshing procedure is simple and efficient (what will be shown later).

Leakage-resilient identification scheme security game

Parameters: n – security parameter, λ – leakage parameter.

Setup stage: $\mathtt{params} \leftarrow \mathsf{ParamGen}(1^n)$; $(\mathtt{pk}, \mathtt{sk}) \leftarrow \mathsf{KeyGen}(1^n; R)$; $\mathcal{A}$ gets $\mathtt{pk}, \mathtt{params}$

Test stage: In this stage, $\mathcal{A}^{\mathcal{O}_\lambda^{\mathtt{sk}}, \mathcal{P}(\mathtt{pk},\mathtt{sk})}$ has oracle access to $\mathcal{P}(\mathtt{pk}, \mathtt{sk})$ and learns at $\zeta(n)$ proofs of identity. Furthermore, the adversary has access to the leakage oracle $\mathcal{O}_\lambda^{\mathtt{sk}}$.

Impersonation stage: Having lost access to $\mathcal{P}(\mathtt{pk}, \mathtt{sk})$ and $\mathcal{O}_\lambda^{\mathtt{sk}}$, adversary $\mathcal{A}$ tries to identify herself as prover $\mathcal{P}$ to verifier $\mathcal{V}$. The verifier outputs `Accept` if the adversary succeeds and `Reject` otherwise.

Figure 2.7.: $\mathsf{ID}_{\lambda,n,\zeta(n)}$; $\zeta(n)$-bounded leakage-resilient identification scheme security game.

Hence, we introduce here new notions of security, $\zeta(n)$-bounded leakage-resilient identification and signature schemes, as stated on Figures 2.7 and 2.8. Furthermore, Definition 18 from the section above takes a new, slightly modified, form:

Definition 19 (Entropic unforgeability for a limited number of executions)**.** We say that a signature scheme $(\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ is $(\varepsilon(n), \mathbf{TM})$-*entropically-unforgeable* with leakage parameter λ and number of executions bounded by $\zeta(n)$ if the advantage of any entropic adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2) \in \mathbf{TM}_\lambda^{\mathcal{O}^{\mathtt{sk}}}$ in the game $\mathsf{EUG}_{\lambda,n,\zeta(n)}$ (see Figure 2.8) is not greater than $\varepsilon(n)$.

Similarly to a setting where the adversary can make polynomially many queries to a signature or an identification oracle, when the number of queries is bounded by $\zeta(n)$, signature scheme can be constructed from an identification scheme. Below we formulate a corollary of Theorem 3 that shows security of the construction.

Corollary 4. *Let* $\Pi = (\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$ *by a public coins* $\zeta(n)$*-bounded secure identification scheme consisting of three rounds of interaction initiated by the prover. Let* $\Sigma = (\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Verify})$ *be the signature scheme produced by the Fiat-Shamir paradigm applied to* Π*. If* Π *allows leakage* λ*, then* Σ *is* $\zeta(n)$*-bounded secure entropically-unforgeable with leakage* λ*.*

Proof. The proof goes as the proof of Theorem 3, see Appendix A. For every signature oracle query from adversary $\mathcal{A}$ there is a corresponding identification oracle query from $\mathcal{B}$. Thus both adversaries are bounded by the same parameter $\zeta(n)$. □

2.11. Basic properties of disperser graphs

Let cryptographic key be divided into blocks, called henceforth key blocks. Similarly, say that data used to derive the key is divided into blocks also. Every block in the key

Leakage-resilient signature scheme entropic unforgeability security game
Parameters: n – security parameter, λ – leakage parameter.

Initialization System parameters and keys get set by $\texttt{params} \leftarrow \mathsf{ParamGen}(1^n)$ and $(\texttt{sigk}, \texttt{verk}) \leftarrow \mathsf{KeyGen}(1^n; R)$. Parameters `params` are publicly known. There is signing oracle $\mathcal{S}$ parametrized by key `sigk` and leakage oracle $\mathcal{O}_\lambda^{\texttt{sk}}$. The signing oracle on input $m \in \mathbf{MsgSpace}$ outputs signature σ such that $\mathsf{Verify}(\texttt{verk}; \sigma, m)$ accepts. The verifier gets public parameters along with verification key `verk`. The signing oracle answers up to $\zeta(n)$ queries.

Signing and leakage queries Adversary $\mathcal{A}_1^{\mathcal{O}_\lambda^{\texttt{sk}}, \mathcal{S}^{\texttt{sigk}}}$ is given access to the signing oracle $\mathcal{S}^{\texttt{sigk}}$ and leakage oracle $\mathcal{O}_\lambda^{\texttt{sk}}$. $\mathcal{A}_1$ outputs an arbitrary hint $v \in \{0,1\}^*$.

Post-leakage Entropic adversary $\mathcal{A}_2^{\mathcal{S}^{\texttt{sigk}}}$ is given the hint v and access to the signing oracle $\mathcal{S}^{\texttt{sigk}}$. The output of $\mathcal{A}_2$ is parsed as a message, signature pair (m, σ). Verifier $\mathcal{V}$ outputs `Accept` if $\mathsf{Verify}(\texttt{verk}; \sigma, m)$ accepts.

Figure 2.8.: $\mathsf{EUG}_{\lambda,n,\zeta(n)}$, $\zeta(n)$-bounded leakage-resilient signature scheme security game

depends on a number of chosen data blocks. We claim that even if the adversary learns a part of data blocks it does not help her predicting the value of a particular key block. This holds since the number of data blocks needed to produce a single key block is big. Furthermore, there is negligible probability that a small number of the learnt data blocks gives any valuable knowledge on a considerable number of key blocks. These claims are supported by the following property of our scheme: to learn a considerable number of key blocks, one has to learn many, almost all, data blocks.

Functions that have the properties expressed above can be delivered from disperser graphs that are described in this section. Informally, we can describe a disperser as a bipartite graph where every small (but big enough) set of vertices from one side has a neighborhood consistent of almost all vertices on the other.

We do not claim originality of the upcoming considerations – for instance, similar arguments appear in [HLW06] or Vadhan's survey [Vad12]. Despite this, we were not able to find the results of presented below Corollary 8 in literature. In the following U_N describes a random variable of uniform distribution over $\{0,1\}^N$.

Before we proceed with disperser graphs, we recall a few elementary definitions from the graph theory.

Definition 20 (Neighborhood, see [BCN89])**.** Let $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ be an undirected graph with a set of vertices $\mathbf{V}$ and edges $\mathbf{E}$. For a subset $\mathbf{S} \subset \mathbf{V}$ we denote by $N(\mathbf{S})$ set $\{w \in \mathbf{V} : \exists s \in \mathbf{S}, (s, w) \in \mathbf{E}\}$, i.e. the set of all neighbors of $\mathbf{S}$, and call it neighborhood of set $\mathbf{S}$ in graph $\mathcal{G}$.

Since disperser is a bipartite graph, we introduce the following definitions.

Definition 21 (Bipartite graph, see [Die12])**.** We call a graph $\mathcal{G} = (\mathbf{V}, \mathbf{E})$ bipartite if

there exist two subsets $\mathbf{V}^0 \sqcup \mathbf{V}^1 = \mathbf{V}$ such that for each $v \in \mathbf{V}^b$, $b \in \{0,1\}$ holds:

$$(v,w) \in \mathbf{E} \implies w \in \mathbf{V}^{1-b};$$

i.e. every node in $\mathbf{V}^0$ shares edges only with nodes from $\mathbf{V}^1$ and vice-versa. We call set $\mathbf{V}^0$ left nodes, and $\mathbf{V}^1$ right nodes.

Definition 22 (Bipartite graph regularity, see [Die12])**.** We say that a bipartite graph $\mathcal{G} = (\mathbf{V}^0 \sqcup \mathbf{V}^1, \mathbf{E})$ is *right d-regular* if all vertices in $\mathbf{V}^1$ have the same degree d.

Definition 23 ((κ,ε)-extractor, see [AB09])**.** We say that a function $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^d \to \{0,1\}^m$ is an (κ,ε)-extractor if for any random variable X of min-entropy $\mathrm{H}_\infty(X) \geqslant \kappa$ and S uniformly distributed over $\{0,1\}^d$ we have $\Delta(\mathrm{Ext}(X,S), U_m) \leqslant \varepsilon$.

Definition 24 (Expander graph, see [Ta-02])**.** A bipartite graph $\mathcal{G} = (\mathbf{V}^0 \sqcup \mathbf{V}^1, \mathbf{E})$ is a *right (K,L)-expander* if for every set $\mathbf{S} \subset \mathbf{V}^1$ such that $|\mathbf{S}| = K$ the neighbourhood $N(\mathbf{S})$ satisfies

$$|N(\mathbf{S})| \geqslant L,$$

i.e. the sets of size K expands into sets of size at least L.

We shall use the following simple lemma which describes expansion properties of bipartite expander graphs.

Lemma 5. *Let $\mathcal{G} = (\mathbf{V}^0 \sqcup \mathbf{V}^1, \mathbf{E})$ be a bipartite right (K,L)-expander. Then for every set $\mathbf{S} \subset \mathbf{V}^1$ such that $|\mathbf{S}| \geqslant K$ and every subset $\mathbf{T} \subset \mathbf{V}^0$ of size $|\mathbf{T}| < L$ the set $N(\mathbf{S})$ is not contained in $\mathbf{T}$.*

Proof. This is a direct counting. Take $\mathbf{S}' \subset \mathbf{S}$ of size equal to K. The size of $N(\mathbf{S})$ then satisfies:

$$|N(\mathbf{S})| \geqslant |N(\mathbf{S}')| \geqslant L.$$

which finishes the proof. □

A special case of expander graphs are dispersers, which are defined as below.

Definition 25 (Disperser graph, see [Ta-02])**.** A bipartite graph $\mathcal{G} = (\mathbf{V}^0 \sqcup \mathbf{V}^1, \mathbf{E})$ is a *right (K,η)-disperser* if for every set $\mathbf{S} \subset \mathbf{V}^1$ such that $|\mathbf{S}| = K$ the neighbourhood $N(\mathbf{S})$ satisfies

$$|N(\mathbf{S})| \geqslant (1-\eta)|\mathbf{V}^0|,$$

i.e. the sets of size K expands into sets of size at least $(1-\eta)|\mathbf{V}^0|$.

In order to instantiate the above considerations, we will use the following correspondence between dispersers and randomness extractors. Observe that every function $f : \{0,1\}^n \times \{0,1\}^m \to \{0,1\}^n$ corresponds to a right 2^m-regular bipartite graph $\mathcal{G}_f = (\mathbf{V}_f, \mathbf{E}_f)$ defined

by:

$$\mathbf{V}_f = \mathbf{V}_f^0 \sqcup \mathbf{V}_f^1 \text{ for } \mathbf{V}_f^0 = \mathbf{V}_f^1 = \{0,1\}^n \tag{2.4}$$

$$\mathbf{E}_f = \{(v_0, v_1) : \exists e \in \{0,1\}^m, f(v_1, e) = v_0\}. \tag{2.5}$$

It is worth reminding that a graph defined this way is in fact a multigraph, i.e. a graph that two vertices can be connected by more than one edge. It turns out that randomness dispersing properties of f are closely related to the vertex expansion of $\mathcal{G}_f$. Namely, the following theorem holds.

Theorem 6. *(see [Vad12], Proposition 6.20) Let $D : \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^n$ be a function such that for any random variable $X \in \{0,1\}^n$ of min-entropy $H_\infty(X) \geqslant \delta n$ and E distributed uniformly on $\{0,1\}^s$, the statistical distance $\Delta(D(X,E), U_n)$ is less than or equal to ε. Then for every $\mathbf{S}$, a subset of right side of a bipartite graph $\mathcal{G}_D = (\mathbf{V}_D^0 \sqcup \mathbf{V}_D^1, \mathbf{E}_D)$ of size $2^{\delta n}$ the neighbourhood $N(\mathbf{S})$ satisfies*

$$|N(\mathbf{S})| \geqslant (1-\varepsilon) \cdot 2^n,$$

i.e. the graph $\mathcal{G}_D$ is a right 2^s-regular $\left(2^{\delta n}, \varepsilon\right)$-disperser.

Proof. Take $\mathbf{S}$ a subset of $\{0,1\}^n$ of size $2^{\delta n}$, $X_{\mathbf{S}}$ a random variable distributed uniformly on $\mathbf{S}$ and E distributed uniformly on $\{0,1\}^s$. Observe that $\mathrm{H}_\infty(X_{\mathbf{S}}) \geqslant \delta n$ and therefore, by assumptions concerning D, the inequality $\Delta(D(X_{\mathbf{S}}, E), U_n) \leqslant \varepsilon$ holds. Moreover, we see that $\Pr(D(X_{\mathbf{S}}, E) = S) \neq 0$ exactly for $S \in N(\mathbf{S})$, so $\Delta(D(X_{\mathbf{S}}, E), U_n) \geqslant (2^n - |N(\mathbf{S})|) \cdot \frac{1}{2^n}$. Combining these two inequalities we obtain

$$\varepsilon \geqslant \Delta(D(X_{\mathbf{S}}, E), U_n) \geqslant (2^n - |N(\mathbf{S})|) \cdot \frac{1}{2^n},$$

which is equivalent to the proposition. □

The existence of an efficiently computable function Ext (we require efficient computability in order to instantiate $\mathcal{G}_{\mathrm{Ext}}$ effectively), satisfying the assumptions of Theorem 6 follows from the result proven in [GUV09]:

Theorem 7 (Theorem 1.5 in [GUV09])**.** *For every constant $\alpha > 0$ and all positive integers n, k and all $\varepsilon > 0$, there is an explicit construction of a (k,ε)-extractor* $\mathrm{Ext} : \{0,1\}^n \times \{0,1\}^s \to \{0,1\}^m$ *with $s = O(\log n + \log(1/\varepsilon))$ and $m \geqslant (1-\alpha)k$.*

More precisely, for any $\eta > 0$ we take an $(n(1+\eta), \varepsilon)$-extractor $\mathrm{Ext} : \{0,1\}^{(1+\gamma+\eta)n} \times \{0,1\}^s \to \{0,1\}^n$ which corresponds to the choice $\alpha = \frac{\eta}{1+\eta}$ in above Theorem 7, and interpret it as a function $\widetilde{\mathrm{Ext}} : \{0,1\}^n \times \{0,1\}^{(\gamma+\eta)n+s} \to \{0,1\}^n$. Then, taking $X \in \{0,1\}^n$ of min-entropy $\mathrm{H}_\infty(X) \geqslant (1-\gamma)n$ we see that $\Delta(\widetilde{\mathrm{Ext}}(X, U_{(\gamma+\eta)n+s}), U_n) \leqslant \varepsilon$ and therefore we may apply Theorem 6 to obtain a $2^{(\gamma+\eta)n+s}$-regular $(2^{(1-\gamma)n}, \varepsilon)$-disperser. As η might be chosen arbitrarily and $s = O(\log n + \log(1/\varepsilon))$ we obtain:

Corollary 8. *For any $\varepsilon > 0$ and $c > 0$, there exists $n_c \in \mathbb{N}$ such that for $n > n_c$ and (d, k) such that $d \cdot k \geqslant 2^{(1+c)n}$, there exists an effectively computable bipartite right d-regular graph which is a right (k, ε)-disperser. In particular, for any $\alpha > \frac{1}{2}$ and $\ell = 2^n$ sufficiently large there exists a bipartite right ℓ^α-regular $(\sqrt{\ell}, \varepsilon)$-disperser.*

Proof. Apply Theorem 6 and the above discussion after Theorem 7. □

CHAPTER 3

DISPERSE AS A KEY DERIVATION FUNCTION

In this chapter we present the main result of the thesis. As stated in Section 1.3.2, one of the main drawbacks of the Bounded Retrieval Model is its inevitable space inefficiency. This particular obstacle is especially pinching when a BRM scheme is to be run on devices with limited memory like smartphones or tablets, devices where storage is usually filled with multimedia like photos, videos, music and applications and that do not have enough spare space.

Our main idea is to retreat from storing a separate large BRM key in favour of deriving it from data already stored on a device on the fly. We recall a cryptographic primitive, *key derivation function*, kdf originally introduced in [DY13], that produces a secure cryptographic key for a BRM protocol from a source of imperfect randomness. We claim that any BRM protocol Prot, with a key generated by some KeyGen that can be evaluated by $\mathcal{P}$ internally, can be transformed into space-efficient protocol Prot$'$. The modified protocol would be such that its secret key comes from kdf. Furthermore, the security and complexity parameters of Prot$'$ and Prot differ only negligibly.

3.1. Key Derivation Function (kdf)

Given defined in Section 2.1 security games, we are ready to formulate precise definitions for privacy and security we impose on our key derivation function.

3.1.1. Privacy of key derivation functions

The following definition captures privacy of kdf. This property is the main difference from an original proposal of kdfs formulated in [DY13], where the authors did not considered such functionality. Privacy is crucial for our idea of deriving a key from user's data,

because a solution which compromises user's privacy cannot be accepted. Moreover, we point out that some BRM protocols allow third parties to learn a part of the key. We can only imagine what could happened if we use non-private kdf on personal data allowing third parties to learn our secrets.

The definition of privacy can be explained as follows: we have two worlds – ideal and real. In the latter, there is an adversary who is given a real key K produced by kdf. In the former, there is a simulator who does not have any access to the key, but knows adversary's code (as a Turing machine). Both parties are equipped with a number of queries to a random and a leakage oracle. We observe outputs from both parties and are given D (i.e. private data) that has been used to produce the key by kdf.

Definition of privacy states that if both worlds are indistinguishable even given D then kdf is private. As in Section 2.2 we argue that this definition is meaningful since it implies that a machine without access to the key, here the simulator, can perform as well as a machine equipped with it, here the adversary. Thus, having access to the key does not give the adversary anything about data D that could not be simulated given public parameters and access to random and leakage oracles. This intuitions were illustrated on Figure 3.1.

Using simulators to show (different flavours of) privacy is well motivated in the cryptographic literature. First of all, an example from Section 2.2 shows how to use simulators to claim the security of an encryption scheme. We show there that the simulator is able to produce a number of encrypted messages of some random text that are indistinguishable from encryption of messages prepared by the adversary. Thus, we conclude that the encryption scheme is secure, since it hides plaintexts well.

Another example how simulation is used to provide privacy is zero knowledge (see, e.g. [Gro06, Gro04]). In zero knowledge proofs a prover is given x belonging to some NP language $\mathcal{L}$ and the corresponding witness w. The prover prepares a proof π showing that $\mathsf{x} \in \mathcal{L}$ without revealing w. On the other side there is a verifier that equipped only with x that verifies proof π. An honest prover convinces an honest verifier with probability 1. The zero knowledge property ensures that even if the verifier is malicious, she does not learn any information about w from proof π. To show that a proof is zero-knowledge we construct a simulator that for a given statement x produce a valid proof π' *without* knowing the corresponding witness w. The claim is (as usual): if the adversary cannot distinguish whether proof she is given comes from the simulator or the real prover then she learns nothing about the witness from the given proof. See Section 2.2 for more detailed explanation on the ideal and the real world.

Definition 26 (Privacy of a key derivation function)**.** We say that a randomized function $\mathsf{kdf}_{\mathcal{H}} : \mathbb{N} \times \{0,1\}^N \to \{0,1\}^M$ is $(\Lambda(\lambda), q(n), \varepsilon(n))$-private for sources of min-entropy at least κ if there exists simulator $\mathcal{S} \in \mathbf{TM}^{\mathcal{O}^{D,\mathcal{H}}}_{\Lambda(\lambda),q(n)}$ such that for every random variable $D \in \{0,1\}^N$ of min-entropy $\mathrm{H}_\infty(D) \geqslant \kappa$ and every adversary $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}^{D,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}$ operating

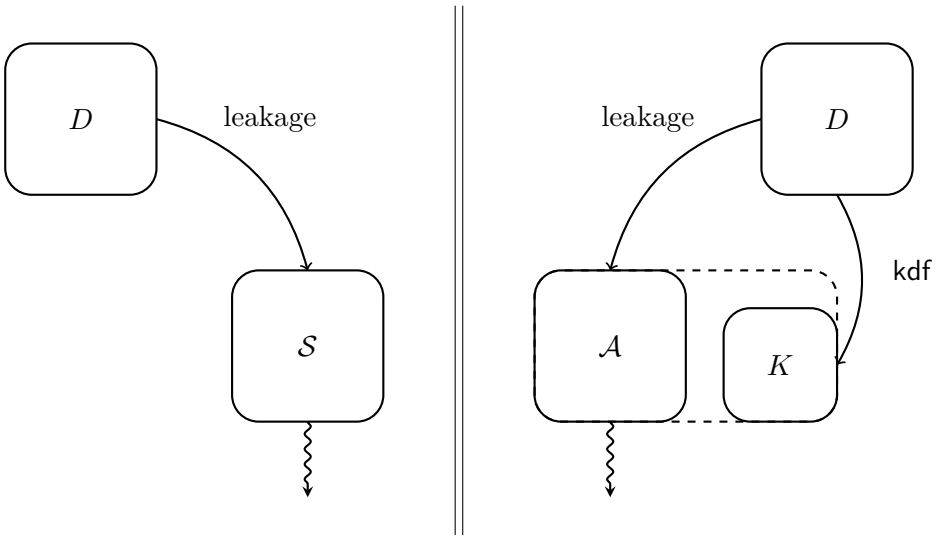


Figure 3.1.: Intuitions behind the definition of privacy. On the right, key K is produced by kdf and given to adversary $\mathcal{A}$. Wavy arrows denote outputs of algorithms. We claim that these outputs are indistinguishable even if D is known for the distinguisher.

on $K \leftarrow \mathsf{kdf}_{\mathcal{H}}(1^n, D)$, the output distributions satisfy:

$$\Delta((\mathbf{Output}(\mathcal{A}(K)), D), (\mathbf{Output}(\mathcal{S}(\mathcal{A})), D)) \leqslant \varepsilon(n).$$

The privacy definition tracks down the amount of additional leakage $(\Lambda(\lambda) - \lambda)$ (denoted by Δ_λ) that is necessary to construct $\mathcal{S}$ capable of simulating the behaviour of any adversary $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}^{D,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}$ (cf. Section 2.4 for the formal specification of this class of algorithms) operating on the key generated by the dispersing procedure. Observe that any algorithm $\mathcal{A}$ is provided with access to an oracle $\mathcal{H}$, i.e. she can query a random function, and moreover she can issue a sequence of leakage queries $\mathcal{O}^{D,\mathcal{H}}(f_i)$, which may also depend on the oracle $\mathcal{H}$, thus can learn some information concerning D depending on the same random function $\mathcal{H}$.

3.1.2. Security of key derivation functions

The next definition explains the meaning of security in terms of kdf. Let us have some BRM protocol Prot with security defined by a security game. Assume Prot is $(\varepsilon(n), \mathbf{TM})$-secure for a security parameter n and function ε, for a key taken uniformly at random. Then Prot is $(\varepsilon(n) + \varepsilon'(n), \mathbf{TM}')$-secure, for some function ε' and key derived by a kdf. In such a case we say that kdf is $(\varepsilon'(n), \mathbf{TM}')$-secure.

The general aim of the security definition is to grasp the intuitive expectation that an adversary playing against key derived from sensitive data should not gain any non-negligible advantage comparing to the case of using a truly random key.

Definition 27 (Security of key derivation function)**.** We say that a randomized function $\mathsf{kdf}_{\mathcal{H}} : \mathbb{N} \times \{0,1\}^N \to \{0,1\}^M$ is $(\Lambda(\lambda), q(n), \varepsilon'(n))$-secure for sources of min-entropy at least κ if for any $\left(\varepsilon(n), \mathbf{TM}^{\mathcal{O}^{K,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}\right)$-secure game

$$\mathsf{Game} = (\mathcal{C}, \mathtt{params} \leftarrow \mathsf{ParamGen}(1^n), K \leftarrow \mathsf{KeyGen}(1^n; R), \mathsf{Setup}_{\mathcal{C}}, \mathsf{Setup}_{\mathcal{A}}, \mathsf{Execute})$$

for randomness R of uniform distribution over $\{0,1\}^N$ and random variable D of length N and min-entropy at least κ, the game

$$\mathsf{Game}_{\mathrm{Disk}} = (\mathcal{C}, \mathtt{params} \leftarrow \mathsf{ParamGen}(1^n), K' \leftarrow \mathsf{kdf}_{\mathcal{H}}(1^n, D), \mathsf{Setup}_{\mathcal{C}}, \mathsf{Setup}_{\mathcal{A}}, \mathsf{Execute}) \tag{3.1}$$

based on randomness $(D, \mathcal{H})$ is $\left(\varepsilon'(n) + \varepsilon(n), \mathbf{TM}^{\mathcal{O}^{D,\mathcal{H}},\mathcal{H}}_{\Lambda(\lambda),q(n)}\right)$-secure.

Note that in the definition below we change the adversary. More precisely, the adversary for the original game Game leaks only from key K and oracle $\mathcal{H}$, while the adversary for $\mathsf{Game}_{\mathrm{Disk}}$ can also leak from disk data. Since the data determine the key, the adversary that can leak from the data, can leak from the key also. Unfortunately, the leakage parameter changes slightly from λ to some smaller $\Lambda(\lambda)$.

From now on, D is a random variable representing disk data, D is divided into ℓ blocks each of length n. λ denotes the number of bits the adversary can leak and p is the ratio of actual min-entropy $\mathrm{H}_\infty(D)$ and the length of D, i.e. $p = \frac{\mathrm{H}_\infty(D)}{n\ell}$

3.2. Disperse **graph**

With kdfs defined, we build a concrete example of a key derivation function. It is based on bipartite disperser graphs. Throughout the whole construction we make use of bipartite right d-regular graph $\mathcal{G}_\sigma$ identified with function

$$\sigma : [|\mathbf{V}^1|] \times [|\mathbf{E}|] \to [|\mathbf{V}^0|]$$

by the following recipe. By $\mathcal{G}_\sigma$ we denote a bipartite graph $\mathcal{G}_\sigma = (\mathbf{V}^0 \sqcup \mathbf{V}^1, \mathbf{E})$ with the sets of vertices equal to two disjoint sets $\mathbf{V}^0, \mathbf{V}^1$ and set of edges $\mathbf{E}$ going from $i \in \mathbf{V}^1$ to $\sigma(i,j) \in \mathbf{V}^0$ for any $j \in [|\mathbf{E}|]$.

We often make use of explicit ℓ^δ-regular (ℓ^e, η)-disperser. We implicitly assume that the numbers δ, e satisfy $\delta < 1, e < 1$ and $\delta + e > 1$. For shorter notation we take $d = \ell^\delta$. For more details on dispersers and further definitions see Section 2.11.

Disperse function takes as parameters $\mathcal{G}_\sigma$ and $\mathcal{H}$. The former is a bipartite graph defined by function σ as above. The latter is a random oracle $\mathcal{H}\colon \{0,1\}^{dn+\log \ell} \to \{0,1\}^n$. While function σ can be passed to Disperse explicitly as a set of argument-value pairs, domain and range size of oracle $\mathcal{H}$ are too large ($2^{dn+\log \ell} + 2^n$ elements together) to be handled the same way. Thus, we assume that Disperse has an oracle access to $\mathcal{H}$ and time needed to

compute $\mathcal{H}(x)$ for given x is constant and does not affect overall computational complexity of Disperse.

The function takes as input a bitstring of length $n\ell$ and outputs a bitstring of the same length. According to our application we identify the input with user's private data that are used to derive cryptographic key. Having parameters and input specified, Disperse works as described in Figure 3.2. An exemplary Disperse function is shown in Figure 3.3.

Implementation of $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D)$**.**

Parameters:

1. d-regular bipartite right (ℓ^e, η)-disperser $\mathcal{G}_\sigma = (\mathbf{V}^0 \sqcup \mathbf{V}^1, \mathbf{E})$ defined by function $\sigma : [|\mathbf{V}^1|] \times [|\mathbf{E}|] \to [|\mathbf{V}^0|]$, such that $|\mathbf{V}^0| = |\mathbf{V}^1| = \ell$ and $|\mathbf{E}| = d\ell$.
2. random oracle $\mathcal{H} : \{0,1\}^{dn+\log \ell} \to \{0,1\}^n$.

Input: Bitstring D of length $n\ell$.
Output: Key $D' = D'_1, \ldots, D'_\ell \in \{0,1\}^{n\ell}$.

Execution:

1. Input $D \in \{0,1\}^{n\ell}$ is divided into ℓ, n-bit long strings: $D_1, \ldots, D_\ell$.
2. Output of Disperse is divided into blocks as well. We denote these blocks as $D'_1, \ldots, D'_\ell$.
3. For $i = 1, \ldots, \ell$: output block D'_i, gets value $\mathcal{H}(i, D_{\sigma(i,1)}, \ldots, D_{\sigma(i,d)})$,
4. Disperse returns $D' = D'_1 \ldots D'_\ell$.

Figure 3.2.: Implementation of Disperse function.

We now state our main result that for an appropriately chosen graph $\mathcal{G}_\sigma$ the function $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}$ is in fact private and secure for reasonable parameters.

Theorem 9. *Let $\mathcal{G}_\sigma$ be a d-regular (ℓ^e, η)-right disperser, $D \in \{0,1\}^{n\ell}$ a random variable and $\mathcal{H}$ a random oracle. Then, the function* $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}$ *is:*

- $(\Lambda(\lambda) = \lambda + \ell^e(\log q(n) + \log \ell), q(n) = 2^{o(\ell^{1-e})n}, \varepsilon(n) = 2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)})$-*private*
- $(\Lambda(\lambda) = \lambda - \ell^e(\log q(n) + n), q(n) = 2^{o(\ell^{1-e})n}, \varepsilon(n) = 2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)})$-*secure,*

for $\eta \leqslant p$ and $\mathrm{H}_\infty(D) > pn\ell$.

Since the proof of this theorem is long, it is divided into several parts. At the beginning it is shown that even under the presence of leakage, function Disperse effectively hides

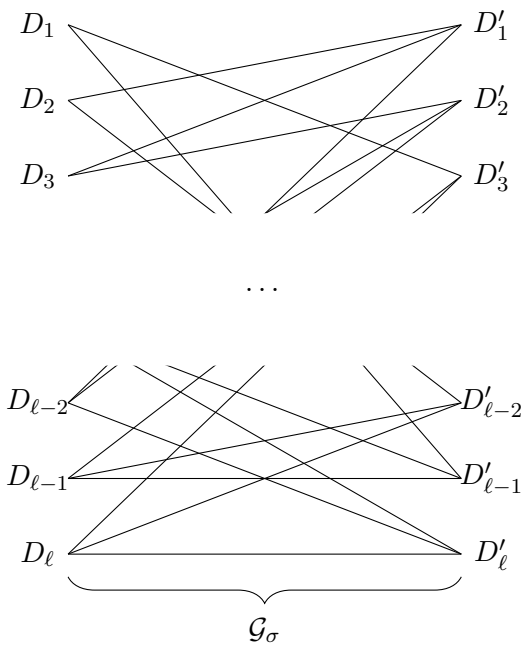


Figure 3.3.: An exemplary $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D)$ built on a 3-regular right disperser $\mathcal{G}_\sigma$.

the data underneath. Then, using this result, privacy property is proven. Later, we proceed with showing security. At the end, we elaborate shortly about the efficiency of $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}$.

Before proceeding to actual proofs we shortly discuss the bounds on parameters.

Remark (Efficiency of $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}$). It is important to note that in order to obtain a single bit of a derived key one need to process d blocks of disk data. Therefore, this constitutes a leakage–efficiency trade-off for the operation of our function. More precisely, reduction of d allows to compute a single bit (and consequently a block) of the key quicker at a cost of an decreased parameter λ proportionally to ℓ^e (recall that $d \cdot \ell^e > \ell$).

Remark (Bounds on parameters). The bound $\eta \leqslant p$ express natural requirements that the quality of disperser η should be superior to the entropy reserve represented by p. The bound on $q(n) = 2^{o(\ell^{1-e})n}$ corresponds to a robust, exponential bound on the random oracle query based complexity of the adversary.

3.3. Guessing game

We start with the definition of a Guessing game which is tailored to be used in the proof of Lemma 11. The idea for the game comes from [DKW11]. In that paper the authors shown similar yet much simpler game. More precisely, they assumed that X_i are pairwise independent random variables with values from $\{0,1\}$. Here we consider X_i from much bigger set $\{0,1\}^n$ and the player additionally equipped with leakage. Informally, this game

shows an upper bound for probability of guessing a value of random oracle $\mathcal{H}$ output for a particular input given some leakage and a number of $(x, \mathcal{H}(x))$ pairs.

Definition 28 (Guessing game)**.** Let $X : \{0,1\}^{n\ell} \to \{0,1\}^{n\ell}, \mathcal{H} : \{0,1\}^{dn+\log \ell} \to \{0,1\}^n$ be random variables. Let $\mathrm{H}_\infty(X) = pn\ell$ for some $0 < p \leqslant 1$. $\mathsf{Guessing}_{X,\mathcal{H}}(p, k_1, k_2, \lambda_{\mathrm{Leak}})$ *game against adversary* $\mathcal{A}$ consists of the steps described in Figure 3.4.

$\mathsf{Guessing}_{X,\mathcal{H}}(p, k_1, k_2, \lambda_{\mathrm{Leak}})$ **game for adversary** $\mathcal{A}$**.**

Input: random variables $(X, \mathcal{H})$, where $X = (X_1, \ldots, X_\ell) \in \{0,1\}^{\ell n}$ and $\mathcal{H} = (\mathcal{H}_{v_1}, \ldots, \mathcal{H}_{v_N}) \in \{0,1\}^{Nn}$ for some parameters ℓ, N and labels $v_i \in \{0,1\}^n$. (The values of X_i should be considered as certain $\mathcal{H}$ labels.) Furthermore, we declare k_1, k_2, leakage parameter λ_{Leak} and p such that X has min-entropy at least $p\ell n$.

Leakage phase:

1. $\mathcal{A}$ issues a leakage query $\mathrm{Leak}(X_1, \ldots, X_\ell, \mathcal{H})$ of length λ_{Leak} and learns corresponding values.

First phase:

2. $\mathcal{A}$ adaptively queries $\mathcal{H}$ by submitting a label v and receiving $\mathcal{H}_v$.
3. $\mathcal{A}$ chooses a subset of indices $\mathbf{S}_1 \subset [\ell]$ of size k_1 along with the guesses for all values $\{X_j \mid j \in \mathbf{S}_1\}$.

Second phase:

4. $\mathcal{A}$ receives all values of $\{X_i \mid i \notin \mathbf{S}_1\}$; that is all blocks that she did not try to guess.
5. $\mathcal{A}$ outputs a subset of labels $\mathbf{S}_2 \subset \{v_i\}_{i=1,\ldots,N}$ of size k_2 which were not previously (i.e., in the first phase) queried along with guesses for all values $\{\mathcal{H}_v \mid v \in \mathbf{S}_2\}$.

Figure 3.4.: Definition of Guessing game

Lemma 10. *Let* $(X_1, \ldots, X_\ell, \mathcal{H})$ *be a random variable such that:*

1. $(X_1, \ldots, X_\ell)$ *and* $\mathcal{H}$ *are independent,*
2. $\mathcal{H}$ *is a vector of random independent* $N = 2^{dn+\log \ell}$ *blocks of length* n*,*
3. *each* X_i *is* n *bits long,*
4. $\mathrm{H}_\infty(X_1, \ldots, X_\ell) = p\ell n$ *for some* $p \in (0, 1]$ *.*

Let $\mathcal{A}$ be a randomized algorithm playing Guessing *game with leakage λ_{Leak}. Then, the probability that $\mathcal{A}$ outputs all correct guesses in both phases is at most*

$$2^{-n(k_1+k_2-(1-p)\ell)+\lambda_{\mathrm{Leak}}} ,$$

for $k_1 \geqslant (1-p)\ell$.

Proof. Suppose there exists algorithm $\mathcal{A}$ winning the game with probability α. This implies that $\alpha \leqslant 2^{\lambda_{\mathrm{Leak}}} \cdot \beta$, where β is an upper bound for probability of winning for any non-leakage algorithm $\mathcal{B}$. The inequality holds since $\mathcal{B}$ can simply guess λ_{Leak} bits of leakage. Algorithm $\mathcal{B}$ does so with probability at least $2^{-\lambda_{\mathrm{Leak}}}$, thus if $\alpha > 2^{\lambda_{\mathrm{Leak}}} \cdot \beta$ then $\mathcal{B}$ could come up with a strategy that gives better probability of winning. It would just run $\mathcal{A}$ and guess her leakage. This strategy would give $\mathcal{B}$ probability of winning at least $\alpha \cdot 2^{-\lambda_{\mathrm{Leak}}} > \beta$ and contradicts the assumption that $\mathcal{B}$ wins with probability at most β.

Suppose that β_1 is probability of guessing k_1 values from $\mathbf{S}_1$ for some non-leakage algorithm $\mathcal{B}$. The probability of guessing the whole X is at least $\beta_1 \cdot 2^{-(\ell-k_1)n}$, since $\mathcal{B}$ can guess the missing bits. On the other hand, by definition of min-entropy, this probability is not greater than $2^{-\mathrm{H}_\infty(X)}$, thus

$$\begin{aligned} \beta_1 \cdot 2^{-(\ell-k_1)n} &\leqslant 2^{-pn\ell} \\ \beta_1 &\leqslant 2^{-n(p\ell-\ell+k_1)} . \end{aligned}$$

Probability β_2 of guessing the second phase of the game by an algorithm $\mathcal{B}$ can be bounded by the same inequality, so

$$\beta_2 \leqslant 2^{-nk_2} .$$

Since X and $\mathcal{H}$ are independent, the probability β that $\mathcal{B}$ wins both phases, is not greater than $2^{-n(p\ell-\ell+k_1+k_2)}$. As it was stated before, leakage λ_{Leak} can be simple guessed and $\alpha \leqslant 2^{\lambda_{\mathrm{Leak}}} \cdot \beta$, hence

$$\alpha \leqslant 2^{-n(k_1+k_2-(1-p)\ell)+\lambda_{\mathrm{Leak}}} .$$

□

3.4. One-wayness of Disperse

The Disperse function possesses a certain *one-wayness* property expressed in Lemma 11. This property is crucial for both security and privacy of the function since it shows that no meaningful information on function's input can be retrieved from the output. We precede the definition of one-wayness introducing *bad queries*:

Definition 29 (Bad query)**.** Given a random variable D, a bipartite right d-regular graph $\mathcal{G}_\sigma$ and a random oracle $\mathcal{H}$ we say that a random oracle query $\mathcal{H}(b)$, submitted by an algorithm $\mathcal{A}$, is *bad* if the argument b equals $(i, D_{\sigma(i,1)}, D_{\sigma(i,2)}, \ldots, D_{\sigma(i,d)})$ for some $i \in$

$\{1, \ldots, \ell\}$, i.e. the argument of a random oracle query equals one of the arguments defined by graph $\mathcal{G}_\sigma$ and a random variable D.

By $\mathbf{Bad}_{\mathcal{A}}$ we denote the set of all bad queries submitted by $\mathcal{A}$. By $\mathbf{indices}_{\mathcal{A}}$ we denote a list of all pairs (k, i_k) of indices $k \in \{1, \ldots, q(n)\}$ and $i_k \in \{1, \ldots, \ell\}$ such that k is the smallest index of a bad query submitted by $\mathcal{A}$ equal to $\left(i_k, D_{\sigma(i_k,1)} \ldots D_{\sigma(i_k,d)}\right)$. Since the total number of queries is bounded by $q(n)$ and $\mathcal{G}_\sigma$ has 2ℓ vertices, we can describe the list $\mathbf{indices}_{\mathcal{A}}$ using $|\mathbf{indices}_{\mathcal{A}}| \cdot (\log \ell + \log q(n))$ bits.

Lemma 11 (One-wayness of $\mathsf{Disperse}$)**.** *Let $\mathcal{G}_\sigma$ be a d-regular (ℓ^e, η)-right disperser and $D = (D_1, \ldots, D_\ell) \in \{0,1\}^{n\ell}$ be a random variable of min-entropy $p\ell n$. Then, the probability that an algorithm $\mathcal{A}(\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D)) \in \mathbf{TM}^{\mathcal{O}^{D,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}$ makes at least ℓ^e different bad queries satisfies:*

$$\Pr(|\mathbf{indices}_{\mathcal{A}}| \geqslant \ell^e) \leqslant 2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)}$$

for $\eta \leqslant p$.

Proof. The general idea of the proof goes as follows. Assume adversary $\mathcal{A}$ such that her associated list $\mathbf{indices}_{\mathcal{A}}$ is longer or equal to ℓ^e with probability greater than $\xi(n) = 2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)}$. With such $\mathcal{A}$ we construct a player $\mathcal{P}_{\mathcal{A}}$ in game $\mathsf{Guessing}_{(X_1,\ldots,X_\ell),\mathcal{H}}(p, k_1, k_2, \lambda_{\mathrm{Leak}})$ for

$$\begin{aligned}
&(X_1, \ldots, X_\ell) = (D_1, \ldots, D_\ell);\\
&k_1 = (1-\eta)\ell\\
&k_2 = \ell\\
&\lambda_{\mathrm{Leak}} = \lambda + \ell n + \ell^e(\log q(n) + \log \ell),
\end{aligned}$$

winning with probability greater $2^{-n(k_1+k_2-(1-p)\ell)+\lambda_{\mathrm{Leak}}}$, what gives a contradiction to the bound shown in Lemma 10. The detailed construction of $\mathcal{P}_{\mathcal{A}}$ is described in Figure 3.5.

The first step is to show that player $\mathcal{P}_{\mathcal{A}}$ follows the rules of the game:

$$\mathsf{Guessing}_{(D_1,\ldots,D_\ell),\mathcal{H}}\left(p, (1-\eta)\cdot\ell, \ell, \lambda + \ell n + \ell^e(\log q(n) + \log \ell)\right).$$

For this sake, we show that:

- The length of hint $\lambda_{\mathrm{Leak}} = |\mathrm{Leak}(D_1, \ldots, D_\ell, \mathcal{H})|$ is not greater than leakage λ of adversary $\mathcal{A}$ along with $n\ell$ bits needed to feed $\mathcal{A}$ and $\ell^e(\log q(n) + \log \ell)$ to handle bad queries. Indeed, $\lambda_{\mathrm{Leak}} = \lambda + \ell n + \ell^e(\log q(n) + \log \ell)$.
- Rules of the game requires that $k_1 \geqslant (1-p)\ell$, which follows from the assumptions.

Now, we need to show that player $\mathcal{P}_{\mathcal{A}}$ guesses correctly:

(a) $(1-\eta)\ell$ elements in the first phase with probability greater than $\xi(n)$ and

Implementation of player $\mathcal{P}_{\mathcal{A}}$.

Learning phase:

Player $\mathcal{P}_{\mathcal{A}}$:

1. Initiates two sets $\mathbf{S}'_1 \leftarrow \emptyset$ and $\mathbf{S}'_2 \leftarrow \emptyset$, which will define elements to guess in the first and the second phase of the Guessing game.

2. Leaks ℓ values

$$(D'_1, \ldots, D'_\ell) = \left(\mathcal{H}(1, D_{\sigma(1,1)} \ldots D_{\sigma(1,d)}), \ldots, \mathcal{H}(\ell, D_{\sigma(\ell,1)} \ldots D_{\sigma(\ell,d)})\right) \ ,$$

   each of length n, from a leakage oracle $\mathcal{O}^{D,\mathcal{H}}$.

3. Initializes adversary $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}^{D,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}$; feeds her with $D'_1, \ldots, D'_\ell$, then submits her to the leakage oracle $\mathcal{O}^a$ to get a hint $\mathrm{Leak}(D, \mathcal{H})$ containing:
   - results of all leakage queries performed by $\mathcal{A}$ (bounded by the leakage parameter λ)
   - list $\mathbf{indices}'_{\mathcal{A}}$ of ℓ^e first items from $\mathbf{indices}_{\mathcal{A}}$ (bounded by $\ell^e (\log q(n) + \log \ell)$).

Game phase:

Adversary $\mathcal{A}$ is now executed with the following recipe for answering to leakage and oracle queries:

Leakage phase $\mathcal{P}_{\mathcal{A}}$ answers to leakage queries using learned $\mathrm{Leak}(D, \mathcal{H})$.

First phase

Querying $\mathcal{H}$. $\mathcal{P}_{\mathcal{A}}$ queries $\mathcal{H}$ only when $\mathcal{A}$ wants to do so. $\mathcal{P}_{\mathcal{A}}$ follows the rules below:

- $\mathcal{P}_{\mathcal{A}}$ answers to random oracle queries honestly. That is, if $\mathcal{A}$ submits x and asks for $\mathcal{H}(x)$, $\mathcal{P}_{\mathcal{A}}$ passes x to oracle $\mathcal{H}$, learns the result and gives it to $\mathcal{A}$. The only exception are queries equal $\left(j, D_{\sigma(j,1)} \ldots D_{\sigma(j,d)}\right)$, for $j = 1, \ldots, \ell$, which were learnt as part of the leakage $\mathrm{Leak}(D, \mathcal{H})$. In such case $\mathcal{P}_{\mathcal{A}}$ passes to $\mathcal{A}$ value D'_j.
- Every time $\mathcal{A}$ issues a bad oracle query $x = \left(i_k, D_{\sigma(i_k,1)}, \ldots, D_{\sigma(i_k,d)}\right)$ which index k appears on the list $\mathbf{indices}'_{\mathcal{A}}$, player $\mathcal{P}_{\mathcal{A}}$ adds elements $D_{\sigma(i_k,j)}$ to her list of guesses $\mathbf{S}'_1$.

Choosing a subset to guess. $\mathcal{P}_{\mathcal{A}}$ guesses values from set $\mathbf{S}'_1$.

Second phase
- Elements of D that were not guessed in the first phase are revealed.
- $\mathcal{P}_{\mathcal{A}}$ sets $\mathbf{S}'_2 \leftarrow \left\{\left(1, D_{\sigma(1,1)} \ldots D_{\sigma(1,d)}\right), \ldots, \left(\ell, D_{\sigma(\ell,1)} \ldots D_{\sigma(\ell,d)}\right)\right\}$.
- $\mathcal{P}_{\mathcal{A}}$ guesses values from set $\mathbf{S}'_2$.

[a] submitting $\mathcal{A}(D'_1, \ldots, D'_\ell)$ to $\mathcal{O}$ means that $\mathcal{O}^{D,\mathcal{H}}$ gets a description of Turing machine realising $\mathcal{A}$ along with input tape containing $D'_1, \ldots, D'_\ell$.

Figure 3.5.: Implementation of a player $\mathcal{P}_{\mathcal{A}}$

(b) ℓ elements in the second phase.

In order to prove (a) we use the fact that every bad query reveals d associated values $D_{\sigma(i,j)}$ and use the properties of dispersers. More precisely, by the assumptions on $\mathcal{A}$, the length of $\mathbf{indices}_{\mathcal{A}}$ is at least ℓ^e with probability greater than $2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)}$. We denote by $\mathbf{indices}'_{\mathcal{A}}$ list $\mathbf{indices}_{\mathcal{A}}$ truncated to the first ℓ^e entries. The neighbourhood of vertices labeled with $\mathbf{indices}'_{\mathcal{A}}$ consists of at least $(1-\eta)\ell$ elements. These elements were revealed by $\mathcal{A}$ making a bad query, thus they are known to $\mathcal{P}_{\mathcal{A}}$, who collects them in set $\mathbf{S}'_1$. ($\mathcal{P}_{\mathcal{A}}$ handles $\mathcal{A}$ queries, thus she knows what $\mathcal{A}$ is asking about).

The claim (b) follows directly from the definition of $\mathrm{Leak}(D,\mathcal{H})$. Namely, $\mathrm{Leak}(D,\mathcal{H})$ contains the values of $\mathcal{H}\left(i, D_{\sigma(i,1)}\dots D_{\sigma(i,d)}\right)$, for $i = 1,\dots,\ell$, which are explicitly prohibited from being queried (Item 2 of the Game phase) and therefore can be guessed in the second phase of operation of $\mathcal{P}_{\mathcal{A}}$. More precisely, $\mathcal{P}_{\mathcal{A}}$ sets $\mathbf{S}'_2 = \left\{\left(1, D_{\sigma(1,1)}\dots D_{\sigma(1,d)}\right),\dots,\left(\ell, D_{\sigma(\ell,1)}\dots D_{\sigma(\ell,d)}\right)\right\}$. Note that values $\left(i, D_{\sigma(i,1)}\dots D_{\sigma(i,d)}\right)$ are known since they were revealed in the first step of the second phase of the game.

Thus, player $\mathcal{P}_{\mathcal{A}}$ wins the first phase of the Guessing game with probability greater than $2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)}$ and the second with probability 1. These bounds give overall winning probability greater than $2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)}$, breaking the upper bound from Lemma 10.

□

3.5. Privacy of Disperse

In this section we show that Disperse is in fact a private key derivation function. The main idea of the proof is application of the one-wayness property together with a careful design of leakage queries. It is important to note that we extensively use our computational model, where we can submit queries that are not computable in polynomial time. On the other hand, we submit such queries for a polynomially unbounded adversary only. More precisely, we submit queries that are as complex as the adversary is. That is, we allow the simulator to submit the adversary (represented as a Turing machine) as a leakage query.

Before we proceed to the main theorem in this part, we define a *twisted function*. This concept clarifies how a random oracle in our scheme is programmed.

Definition 30 (Twisted function)**.** Let f be a function and $L = ((\arg_1, v_1),\dots,(\arg_k, v_k))$ be a list of pairs of an argument $\arg_i$ together with a potential value v_i. We define a *twisted function* $f\{L\}$ to be function whose operation is described as follows:

$$f\{L\}(q) = \begin{cases} v_i & \text{if } q = \arg_i \text{ for some } i \\ f(q) & \text{otherwise.} \end{cases}$$

In particular, given a random variable D, a random oracle $\mathcal{H}$ and a random variable $K = (K_1,\dots,K_\ell) \in \{0,1\}^{\ell n}$, by $\mathcal{H}\{D \xrightarrow{\mathcal{G}_\sigma} K\}$ we denote a random oracle

$\mathcal{H}\left\{\left(D_{\sigma(1,i)}\ldots D_{\sigma(\deg(\mathcal{G}_\sigma),i)}, K_i\right)_{i=1,\ldots,\ell}\right\}$. Observe that if K is uniformly random over $\{0,1\}^{n\ell}$ and independent of $\mathcal{H}$ then the distributions of $\mathcal{H}\{D \xrightarrow{\mathcal{G}_\sigma} K\}$ and $\mathcal{H}$ are the same.

Theorem 12 (Privacy)**.** *Let $\mathcal{G}_\sigma$ be a d-regular (ℓ^e, η)-right disperser and $D = (D_1, \ldots, D_\ell) \in \{0,1\}^{n\ell}$ be a random variable of min-entropy at least $p\ell n$. Then, there exists simulator $\mathcal{S} \in \mathbf{TM}^{\mathcal{O}^{D,\mathcal{H}},\mathcal{H}}_{\lambda+\ell^e(\log q(n)+\log \ell),q(n)}$ such that for every adversary $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}^{D,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}$ operating on the key $K \leftarrow \mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D)$, the output distributions satisfy:*

$$\Delta((\mathbf{Output}(\mathcal{A}(K)), D), (\mathbf{Output}(\mathcal{S}(\mathcal{A})), D)) \leqslant \varepsilon(n)$$

for $\varepsilon(n) = 2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)}$, $q(n) = 2^{o(\ell^{1-e})n}$ and any $\eta \leqslant p$.

In order to give a proof of this theorem, we shall construct a machine $\mathcal{S}$ such that for any adversary $\mathcal{A}(K) \in \mathbf{TM}^{\mathcal{O}^{D,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}$ the result of $\mathcal{S}(\mathcal{A})$ is indistinguishable from $\mathcal{A}(K)$ conditioned on D.

Construction of the simulator The operation of $\mathcal{S}(\mathcal{A})$, based on the description of $\mathcal{A}$, consists of the steps described in Figure 3.6.

Before giving a formal proof of statistical indistinguishability of output distributions, we give some clarifying remarks about consecutive steps of the construction. Firstly, we should emphasize that in Step (2) we crucially use the properties of our leakage model by querying leakage oracle with potentially non-polynomial function simulating whole behaviour of $\mathcal{A}$. Secondly, observe that in Step (2) the simulator leaks only the indices of queries, not their actual arguments as those can be observed during Step (3) of simulation. Thirdly, note that in Step (3a) we do not need to perform any additional leakage apart from the value of f, as $f\{D \xrightarrow{\mathcal{G}_\sigma} K\}$ can be obtained inside the leakage query as in Step (2). Therefore the leakage excess consists merely of the list $\mathbf{indices}_\mathcal{A}$ and consequently the additional leakage the simulator has to be equipped with to be able to simulate correctly is $\Delta_\lambda = |\mathbf{indices}_\mathcal{A}|(\log q(n) + \log \ell)$.

Proof of Theorem 12 (Privacy). We shall now argue that simulator $\mathcal{S}$ constructed in Figure 3.6 satisfies the requirements of Theorem 12 for any adversary $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}^{D,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}$. Concretely, we prove that $\mathcal{S}$ perfectly simulates the execution of any adversary $\mathcal{A}$, unless $|\mathbf{indices}_\mathcal{A}| \geqslant \ell^e$. Therefore, for any adversary $\mathcal{A}$ the output's distribution of $\mathcal{S}(\mathcal{A})$ satisfies:

$$\Delta((\mathbf{Output}(\mathcal{A}(K)), D), (\mathbf{Output}(\mathcal{S}(\mathcal{A})), D)) \leqslant \varepsilon(n),$$

where $\varepsilon(n) = \Pr(|\mathbf{indices}_\mathcal{A}| \geqslant \ell^e)$.

Firstly, note that the execution of $\mathcal{A}$ inside the leakage function ind (see Step (2)) is perfectly equivalent to an honest execution of $\mathcal{A}$ as $\mathcal{H}\{D \xrightarrow{\mathcal{G}_\sigma} K\}$ is distributed equally to

Implementation of simulator $\mathcal{S}$.

1. Simulator $\mathcal{S}$ initializes a random oracle $\mathcal{H}$ and draws a random variable $K = K_1, \ldots, K_\ell$ of uniform distribution over $\{0,1\}^{n\ell}$.

2. $\mathcal{S}$ initializes the random tape of $\mathcal{A}$ to a fixed sequence of uniformly random bits and then queries the leakage oracle with the Turing machine $\mathsf{ind} : \{0,1\}^{n\ell} \to \{0,1\}^*$ which operates as follows:

   Operation of ind:

   Description of the function Simulate the execution of $\mathcal{A}(K)$ step by step with random oracle queries $\mathcal{H}$ substituted with $\mathcal{H}\{D \xrightarrow{\mathcal{G}_\sigma} K\}$. Every time the adversary $\mathcal{A}$ issues a leakage oracle query given by a Turing machine f, simulator $\mathcal{S}$ provides her with a result of a *twisted leakage function* $f\{D \xrightarrow{\mathcal{G}_\sigma} K\}$, i.e., a Turing machine with all random oracle queries substituted with $\mathcal{H}\{D \xrightarrow{\mathcal{G}_\sigma} K\}$.

   Result The list $\mathbf{indices}_\mathcal{A}$. Returns $\mathbf{indices}_\mathcal{A}$ if its length satisfies $|\mathbf{indices}_\mathcal{A}| < \ell^e$, or $\perp$ otherwise.

   Complexity Leakage: upper-bounded by $|\mathbf{indices}_\mathcal{A}|(\log q(n) + \log \ell)$

3. $\mathcal{S}$ executes $\mathcal{A}(K)$ with a previously initialized (see Step (2)) random tape and $\mathcal{H}$ sampled above (see Step (1)), and then runs it step by step with the following exceptions:

   a) When $\mathcal{A}$ issues a leakage query given by a Turing machine f, simulator $\mathcal{S}$ substitutes it with a twisted leakage function $f\{D \xrightarrow{\mathcal{G}_\sigma} K\}$.

   b) $\mathcal{S}$ keeps track of the number k of random oracle queries issued to $\mathcal{H}$ and every time it appears in a pair $(k, i_k) \in \mathbf{indices}_\mathcal{A}$, replaces the value returned by $\mathcal{H}$ with K_{i_k}. Moreover, it stores the arguments a_k of queries appearing in the list $\mathbf{indices}_\mathcal{A}$ and substitutes the value of $\mathcal{H}$ with K_{i_k} every time a_k appears as an argument.

Figure 3.6.: Implementation of the simulator

$\mathcal{H}$. Consequently, the actual simulation given in Step (3) differs from a perfect simulation only by the condition on $|\mathbf{indices}_{\mathcal{A}}|$, as it is perfectly equivalent to the one performed during the simulator's leakage phase. This condition forces the return of $\perp$ instead of appropriate $\mathbf{indices}_{\mathcal{A}}$ with probability $\Pr(|\mathbf{indices}_{\mathcal{A}}| \geqslant \ell^e) \leqslant \varepsilon(n)$. Consequently, we bound $\varepsilon(n)$ by a factor negligible in the security parameters. Directly by applying Lemma 11 for adversary $\mathcal{A}$ we see that:

$$\varepsilon(n) = \Pr(|\mathbf{indices}_{\mathcal{A}}| \geqslant \ell^e) \leqslant 2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)}.$$

This completes the proof. □

3.6. Security of Disperse

Again, based on the one-wayness of Disperse we prove that our function satisfies the security requirements. We have the following:

Theorem 13 (Security)**.** *Let*

$$\mathsf{Game} = (\mathcal{C}, \mathsf{ParamGen}, \mathsf{KeyGen}(1^n; K), \mathsf{Setup}_{\mathcal{C}}, \mathsf{Setup}_{\mathcal{A}}, \mathsf{Execute})$$

be an $(\varepsilon(n), \mathbf{TM}^{\mathcal{O}^{K,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)})$*-secure game based on uniformly random* K*, then for every* d*-regular* (ℓ^e, η)*-disperser* $\mathcal{G}_\sigma$*,* $\eta \leqslant p$ *and* $q(n) = 2^{o(\ell^{1-e})n}$*, the game*

$$\mathsf{Game}_{\mathrm{Disk}} = (\mathcal{C}, \mathsf{ParamGen}, \mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D), \mathsf{Setup}_{\mathcal{C}}, \mathsf{Setup}_{\mathcal{A}}, \mathsf{Execute})$$

based on randomness D *of min-entropy at least* $pn\ell$ *and random oracle* $\mathcal{H}$ *is*

$$\left(\varepsilon(n) + 2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)}, \mathbf{TM}^{\mathcal{O}^{D,\mathcal{H}},\mathcal{H}}_{\lambda-\Delta_\lambda, 2^{o(\ell^{1-e})}}\right) \text{-}secure\ ,$$

for $\Delta_\lambda = \ell^e(\log q(n) + n)$*.*

Before we continue to the formal proof of the theorem, we show some intuitions behind it. We show the theorem by contradiction. Assume there exists adversary $\mathcal{A}'$ that breaks the security of the game $\mathsf{Game}_{\mathrm{Disk}}$ with probability greater than $\varepsilon(n) + 2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)}$. We construct adversary $\mathcal{A}$ that uses $\mathcal{A}'$ as a subroutine to break ε-security of the game Game. The idea of the construction is following. Adversary $\mathcal{A}$ will pick a mock randomness D_{Mock} of the same distribution as D and claim that K is a key output by $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D_{\mathrm{Mock}})$. Recall, that is not true with a great probability. Next, $\mathcal{A}$ will run $\mathcal{A}'$ on D_{Mock}. For this construction to be successful we need to carry $\mathcal{A}'$'s bad queries carefully. More precisely, we will answer the bad queries such with values of the real key K. To do that we will need to reprogram the random oracle $\mathcal{H}$ to make it return elements of K on $\mathcal{A}'$'s bad queries. We show that since $\mathcal{A}'$ is successful in breaking the security of $\mathsf{Game}_{\mathrm{Disk}}$, she will be also successful when run on the mock randomness

Implementation of $\mathcal{A}$.

Input: The adversarial data $\mathsf{Setup}_{\mathcal{A}'}(\mathtt{params}, K)$ on K uniformly distributed over $\{0,1\}^{n\ell}$ (cf. Step (2) in Figure 2.1).

Setup phase:

1. A simulated random variable D_{Mock} is sampled from the distribution $\mathcal{D}$. An efficient data structure **OracleQueryList** for the on-the-fly storage of random oracle $\mathcal{H}$ queries is prepared.

2. An internal version $\mathcal{A}'$ of $\mathsf{Game}_{\mathrm{Disk}}$ adversary is initialized and given

$$\mathsf{Setup}_{\mathcal{A}'}(\mathtt{params}, K) = \mathsf{Setup}_{\mathcal{A}'}\left(\mathtt{params}, \mathsf{Disperse}_{\mathcal{G}_\sigma, \mathcal{H}\{D_{\mathrm{Mock}} \xrightarrow{\mathcal{G}_\sigma} K\}}(D_{\mathrm{Mock}})\right)$$

as input (as in $\mathsf{Game}\left[\mathcal{A}' \leftrightharpoons \mathcal{C}, K = \mathsf{Disperse}_{\mathcal{G}_\sigma, \mathcal{H}\{D_{\mathrm{Mock}} \xrightarrow{\mathcal{G}_\sigma} K\}}(D_{\mathrm{Mock}})\right]$).

Execution phase:

3. Every time $\mathcal{A}$ is provided with a new message $\mathtt{msg}_{\mathcal{A}}$ (including the one initialized with the 0-th message $\perp$), she performs the following steps:

   a) performs $\mathsf{getIndices}$ leakage:

> **$\mathsf{getIndices}$ leakage**
>
> **Result** The list $\mathbf{indices}_{\mathcal{A}'}$ containing pairs (i, v_i) of an index i and a result v_i of $\mathcal{A}'$'s bad random oracle query (cf. Definition 29) conducted during her operation in $\mathsf{Game}\left[\mathcal{A}' \leftrightharpoons \mathcal{C}, K = \mathsf{Disperse}_{\mathcal{G}_\sigma, \mathcal{H}\{D_{\mathrm{Mock}} \xrightarrow{\mathcal{G}_\sigma} K\}}(D_{\mathrm{Mock}})\right]$ when provided with $\mathtt{msg}_{\mathcal{A}}$ before sending the next message.
>
> **Description of the function** Simulate the behaviour of $\mathcal{A}'$ and check whether random oracle queries are bad or not.
>
> **Complexity Leakage**: $|\mathbf{indices}_{\mathcal{A}'}| \cdot (\log q(n) + n)$, **time**: the same as $\mathcal{A}'$ operation. (For the whole execution.)

   b) simulates the behaviour of $\mathcal{A}'$ with the following recipe for answering leakage and random oracle queries:

> **Replying to leakage and random oracle queries**
>
> **Random oracle queries** If the index i appears in one of the pairs in the list $\mathbf{indices}_{\mathcal{A}'}$ leaked above then return v_i otherwise look up **OracleQueryList** and answer with a value from there or a random element drawn from U_n. In any case, add the whole query to **OracleQueryList**.
>
> **Leakage queries** We answer a leakage oracle query f described by a circuit containing oracle $\mathcal{O}^{D_{\mathrm{Mock}}, \mathcal{H}}$ queries by the same circuit containing $\mathcal{O}^{D_{\mathrm{Mock}}, \mathcal{H}\{D_{\mathrm{Mock}} \xrightarrow{\mathcal{G}_\sigma} K\}}$ instead. Note that in order to substitute all bad queries of $\mathcal{H}$ by $\mathcal{H}\{D_{\mathrm{Mock}} \xrightarrow{\mathcal{G}_\sigma} K\}$), we just need to access D_{Mock}, $\mathcal{H}$ and $\mathcal{O}^K$ which are all given to $\mathcal{A}$.
>
> **Complexity Leakage**: same as $\mathcal{A}'$'s, bounded by $\lambda - \ell^e(\log q(n) + n)$. **Time**: same as $\mathcal{A}'$'s up to time necessary for **OracleQueryList** look ups.

   c) if the total leakage equal to $\lambda - \ell^e(\log q(n) + n) + |\mathbf{indices}_{\mathcal{A}'}| \cdot (\log q(n) + n)$ exceeds λ then terminate with $\perp$;

   d) returns the message prepared by $\mathcal{A}'$.

Figure 3.7.: Implementation of $\mathcal{A}$

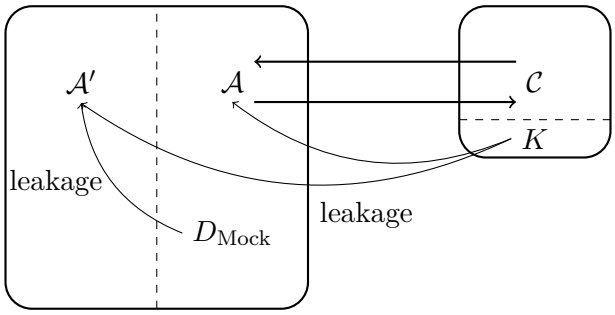


Figure 3.8.: Adversary $\mathcal{A}$ creates fake disk data D_{Mock} and runs kdf-adversary $\mathcal{A}'$, which can leak from D_{Mock} and the key built on it. $\mathcal{A}$ pretends that key K is given by kdf run on D_{Mock}. Thus, when $\mathcal{A}'$ leaks from the kdf key, she in fact leaks from the real key K. When $\mathcal{A}'$ makes a bad query, $\mathcal{A}$ leaks from K and reveals the corresponding key block.

and reprogrammed random oracle. Then $\mathcal{A}$ will be successful in breaking the security of Game. These intuitions were illustrated on Figure 3.8.

Proof. The proof is based on reduction. For sake of contradiction, we assume that there exists an efficiently sampleable random distribution D of min-entropy $pn\ell$ and a $\mathsf{Game}_{\text{Disk}}$-adversary $\mathcal{A}' \in \mathbf{TM}^{\mathcal{O}^{D,\mathcal{H}},\mathcal{H}}_{\lambda-\Delta_\lambda,q(n)}$ such that:

$$\begin{aligned}\Pr_{D\leftarrow\mathcal{D}}(\mathsf{Game}_{\text{Disk}}\left[\mathcal{A}' \leftrightharpoons \mathcal{C}, \mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D)\right] = \texttt{Accept}) &> \varepsilon'(n)\\ &= \varepsilon(n) + \Pr(|\mathbf{indices}_{\mathcal{A}'}| \geqslant \ell^e).\end{aligned}$$

Using $\mathcal{A}'$ as a component, we construct adversary $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}^{K,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}$ contradicting $\left(\varepsilon(n), \mathbf{TM}^{\mathcal{O}^{K,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}\right)$-security of Game based on randomness K. Description of $\mathcal{A}$ is given in Figure 3.7. Note that we shall freely use the twisted random oracles defined in Definition 30.

To finish the proof we need the following claims.

Claim 1 (Simulation). The execution of

$$\mathsf{G} \stackrel{\text{def}}{=} \mathsf{Game}\left[\mathcal{A} \leftrightharpoons \mathcal{C}, \mathsf{KeyGen}(1^n; K)\right]$$

based on randomness K is in fact a simulation of

$$\mathsf{G}' \stackrel{\text{def}}{=} \mathsf{Game}\left[\mathcal{A}' \leftrightharpoons \mathcal{C}, \mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}\{D_{\text{Mock}} \xrightarrow{\mathcal{G}_\sigma} K\}}(D_{\text{Mock}})\right] \ ,$$

since $K = \mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}\{D_{\text{Mock}} \xrightarrow{\mathcal{G}_\sigma} K\}}(D_{\text{Mock}})$ executed on mock randomness $(D_{\text{Mock}}, \mathcal{H})$.

Proof. Firstly, observe that the key K is equal to $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}\{D_{\text{Mock}} \xrightarrow{\mathcal{G}_\sigma} K\}}(D_{\text{Mock}})$ (by definition of $\mathcal{H}\{D_{\text{Mock}} \xrightarrow{\mathcal{G}_\sigma} K\}$) and therefore the input of $\mathcal{C}$ in G is equal to

$\mathsf{Setup}_{\mathcal{C}}\left(\mathtt{params}, \mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}\{D_{\mathrm{Mock}} \xrightarrow{\mathcal{G}_\sigma} K\}}(D_{\mathrm{Mock}})\right)$, for $\mathtt{params} \leftarrow \mathsf{ParamGen}(1^n)$ as in G'.

Moreover, all the messages send by $\mathcal{A}$ are in fact produced by game G' adversary $\mathcal{A}'$ and therefore the only difference is that the leakage and random oracle queries of $\mathcal{A}'$ are not processed honestly but simulated by means of leakage of $\mathcal{A}$ described in Steps (3a) and (3b) in Figure 3.7. □

Claim 2 (Simulation correctness). The simulation above is faithful (i.e., $\mathcal{A}$ works the same as corresponding $\mathcal{A}'$) unless $\perp$ is returned in Step (3c) of simulation. This occurs with probability at most $\Pr(|\mathbf{indices}_{\mathcal{A}'}| \geqslant \ell^e)$ which is upper bounded by $\xi(n) = 2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)}$ and therefore

$$\Pr(\mathcal{A}' \text{ is faithfully simulated}) = 1 - \Pr(|\mathbf{indices}_{\mathcal{A}'}| \geqslant \ell^e) \geqslant 1 - \xi(n).$$

Proof. The first part concerning simulation correctness is clear from the construction. More precisely, the difference between messages of honest $\mathcal{A}'$ and $\mathcal{A}$ occurs only if the restriction λ on the size of leakage (see Step (3c)) intervenes. Therefore we are left to prove that $\Pr(\perp \text{ is returned in (Step 3c)}) \leqslant \xi(n)$. This follows directly by applying Lemma 11 for $\mathcal{A}'$. □

Claim 3 (Leakage bound). The total leakage of $\mathcal{A}$ during the execution of G does not exceed λ bits and consequently $\mathcal{A}$ belongs to $\mathbf{TM}^{\mathcal{O}^{K,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}$.

Proof. In Step (3c) in Figure 3.7 we explicitly stated that $\mathcal{A}$ is terminated if the leakage she makes, that is $\lambda - \ell^e \cdot (\log q(n) + n) + |\mathbf{indices}| \cdot (\log q(n) + n)$, is greater that λ. □

All above claims prove that $\mathcal{A}$ is an efficient adversary for Game which succeeds with probability:

$$\begin{aligned}
\Pr(\mathsf{G} = \mathtt{Accept}) &\geqslant \Pr(\mathsf{G}' = \mathtt{Accept} \text{ and } \mathcal{A}' \text{ is faithfully simulated}) && (3.2)\\
&\geqslant \Pr(\mathsf{G}' = \mathtt{Accept}) + \Pr(\mathcal{A}' \text{ is faithfully simulated}) - 1 && (3.3)\\
&> \varepsilon'(n) + (1 - \xi(n)) - 1 = \varepsilon'(n) - \xi(n) \geqslant \varepsilon(n), && (3.4)
\end{aligned}$$

where in line (3.2) we used (Simulation) claim and in (3.4) we used the fact that the distributions $(D, \mathcal{H})$ and $(D_{\mathrm{Mock}}, \mathcal{H}\{D_{\mathrm{Mock}} \xrightarrow{\mathcal{G}_\sigma} K\})$ are the same and therefore

$$\Pr(\mathsf{G}' = \mathtt{Accept}) = \Pr(\mathsf{Game}[\mathcal{A}' \leftrightharpoons \mathcal{C}, \mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D)] = \mathtt{Accept}) > \varepsilon'(n),$$

and moreover that $\Pr(\mathcal{A}' \text{ is correctly simulated}) \geqslant 1 - \xi(n)$ by Claim (Simulation's correctness).

This contradicts the $\left(\varepsilon(n), \mathbf{TM}^{\mathcal{O}^{K,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}\right)$-security of Game and consequently gives a proof of the theorem as by Lemma 11 the probability $\Pr(|\mathbf{indices}_{\mathcal{A}'}| \geqslant \ell^e) \leqslant 2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)}$. □

3.7. Efficiency of Disperse

Since we defined kdf as an efficiently computable function, the following lemma is necessary to show that cost of computing Disperse is not exaggerated.

Theorem 14 (Complexity of the $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D)$ procedure)**.** *Let $\mathcal{H}$ be a random oracle, $D \in \{0,1\}^{n\ell}$ and $\mathcal{G}_\sigma$ a d-regular right bipartite graph of ℓ vertices in each part. By $K = (K_i)_{i=1}^{n\ell}$, where $K_i \in \{0,1\}$, we denote a key obtained by running $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D)$. Then for any $i \in \{1,\ldots,n\ell\}$ there is needed*

- *read d blocks of D to obtain K_i,*
- *read $2d$ blocks of D to obtain $(K_i,\ldots,K_{i+n-1 \bmod n\ell})$.*

Proof. For any $i \in 1,\ldots,n\ell$ a vertex $D'_i \in D'$ is a neighbour to exactly d vertices from D. Hence, according to Figure 3.2, computing K_i requires to read d blocks of D demanded by the first step of execution of the Disperse function. This finishes the proof of the first part of the statement, the second goes similarly from the observation that any sequence of n consecutive bits from K consists of bits from values assigned to at most two upper vertices of $\mathcal{G}_\sigma$. □

Remark. Assume the key is divided into ℓ blocks of length n each. Then the number of read disk blocks required to obtain the whole key block is d (as in the case of obtaining a single bit of the key).

In signature and identification schemes that are described later, the verifier asks the prover to provide her $k(n)$ blocks of the derived key K. Below we analyse efficiency of such procedure. Note, that we expect efficiency much better than just $k(n) \cdot d$ blocks to read as it could be concluded from the remark above. This holds, since computation of a number of blocks will reuse some queries with high probability.

Theorem 15. *Let $\mathcal{H}$ be a random oracle, $D \in \{0,1\}^{n\ell}$ and $\mathcal{G}_\sigma$ a d-regular bipartite graph of ℓ vertices in each party. By $K = (K_i)_{i=1}^{n\ell}$, where $K_i \in \{0,1\}$, we denote a key obtained by running $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D)$. Let the key be divided into ℓ parts of length n each. The expected value of the number of disk blocks read needed to compute $k(n)$ blocks of the key is*

$$\ell\left(1-\left(1-\frac{d}{\ell}\right)^{k(n)}\right) \quad .$$

Proof. The probability that in $k(n)$ trials a particular block of D is not chosen is $(1-d/\ell)^{k(n)}$. Thus the expected value of the number of blocks that were not chosen is $\ell \cdot (1-d/\ell)^{k(n)}$. Thus, the expected value of the number of chosen blocks is $\ell\left(1-(1-d/\ell)^{k(n)}\right)$. □

3.8. Determining real life parameters

Presented function depends on multiple parameters and may be hard to analyse. Below we show a number of plots to provide intuitions about how *random* disk data and *quality* disperser should be to provide acceptable level of security. We made the following assumptions.

leakage parameter As usually in BRM, we should start with stating how much data information the adversary can leak. Since we assumed that the data is stored on a mobile device, we argue that this leakage bound could be relatively small. Here, we present two results – first, the adversary is able to leak up to 1 GB of data information (see Fig. 3.9, 3.10); second, she is allowed to learn at most 2 GB (see Fig. 3.11, 3.12).

acceptable security We assume that acceptable security level is 2^{-128}. More precisely, in case of *privacy*, statistical distance between the simulator's output and the adversary's one is at most 2^{-128}. In case of *security*, the security loss between protocol run on a truly random key and a key provided by kdf is also bounded by 2^{-128}.

hash function output size In practice, we instantiate a random oracle by a hash function. Since the security parameter equals to the output length of a random oracle, the same holds for a hash function. That is, we set the security parameter to be equal to the hash function output.

The most common hash functions give output of length either 256 or 512 bits. We analyse both cases. We bound the number of hash function queries the adversary can do to 2^{80}. This bound does not restrict adversary much. Single iteration of a hash function takes roughly 12 cycles of the processor [Ber12]. For a hash function output length of 64 bytes and a computer with a 2 GHz processor the adversary can make roughly 2604 hashes per second. That may sound like a lot, yet time needed to compute 2^{80} hashes is still over 10^{13} *years*.

data size Given leakage parameter we set data size. With data size and leakage parameter set we show how *good* the data and disperser have to be. Recall, random variable D has min-entropy $pn\ell$, we say that D is good if p is close to 1. We show the results for a several data sizes.

disperser regularity and quality We need to describe the quality of the disperser. Disperser is defined by two parameters, i.e we called a graph (k, η)-disperser if every set of k nodes on one side is connected with at least $(1 - \eta)$ fraction of nodes on the other. *Good* dispersers have small k and η. Recall, that η can be any number bigger than 0 (however, small η requires bigger number of nodes in the graph). As Corollary 8 suggests, for a (k, η)-disperser of regularity d and ℓ nodes on both sides, it holds $k \cdot d \geqslant \ell$. Thus, the bigger parameter k is, the smaller regularity d could be. Smaller d gives better efficiency, since it means that fewer values of hash function

have to be computed and fewer block data accessed. On the other hand, we cannot set d arbitrary small, since it would make the whole construction insecure.

On the following figures we show how big the difference between the data randomness fraction p and the disperser parameter η, i.e. $(p - \eta)$ (see the vertical axle) should be, given the graph regularity d (see the horizontal axle). Every figure contains a number of plots. Each plot refers to a particular size of disk data. The bigger the data are, the worse quality they could be to achieve desired security level. For given leakage parameter λ (recall, we show results for λ equals 1 GB and 2 GB), we consider disk data of size from 1.1λ (obviously, disk data have to be larger than the leakage) to almost 4λ.

The conclusions from the plots can be delivered as follows.

- The bigger data, the worse quality they could have to deliver desired security.
- Required $(p - \eta)$ quickly emerges to $\frac{\lambda}{|D|}$, usually it is enough to set d around 10 to achieve that.
- Size of the output of the hash function (between 256 and 512) does not change the results much and matters for graphs of small regularity only.

Remark. In case this thesis is printed on a black and white printer. In every figure each label corresponds to the presented plots as follows: the top entry corresponds to the top line, the second to the top corresponds to the second line, etc.

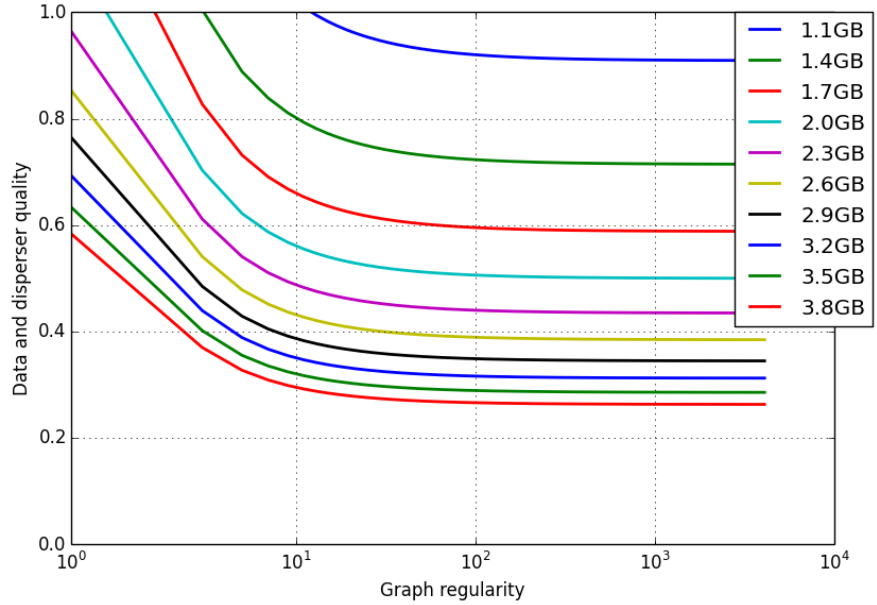


Figure 3.9.: Leakage 1 GB, 256-bit long output of a hash function

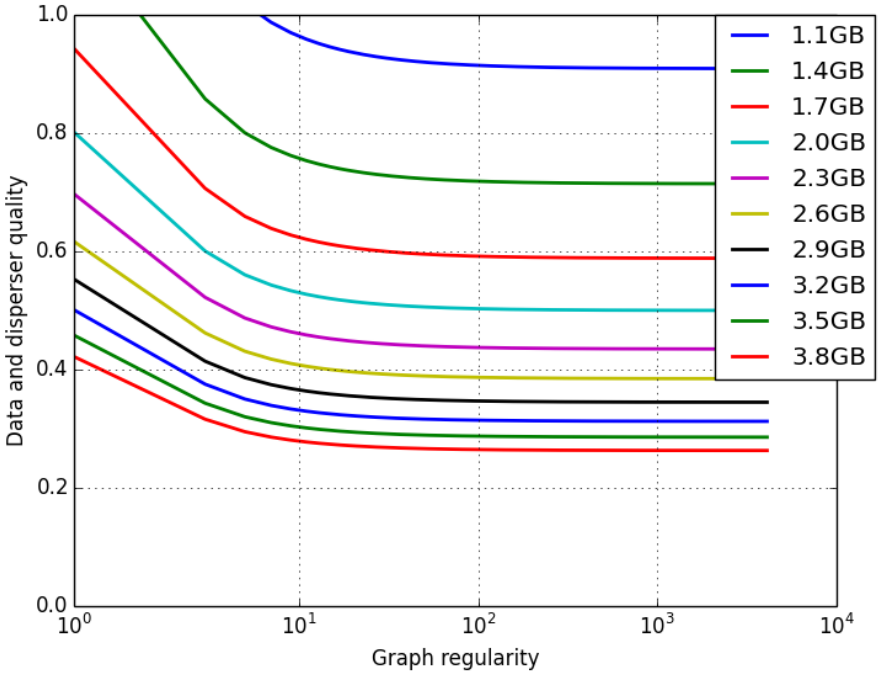


Figure 3.10.: Leakage 1 GB, 512-bit long output of a hash function

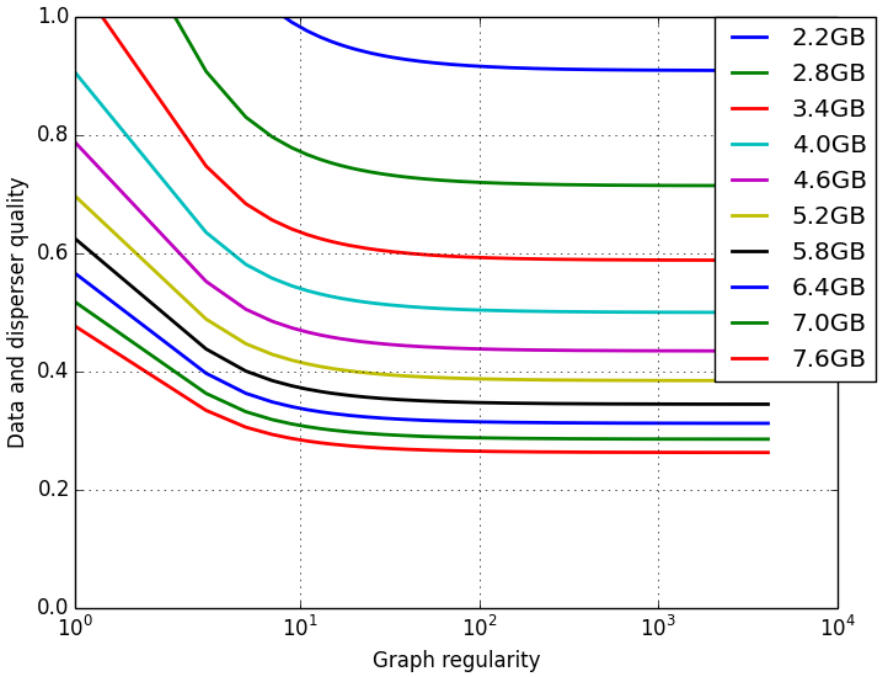


Figure 3.11.: Leakage 2 GB, 256-bit long output of a hash function

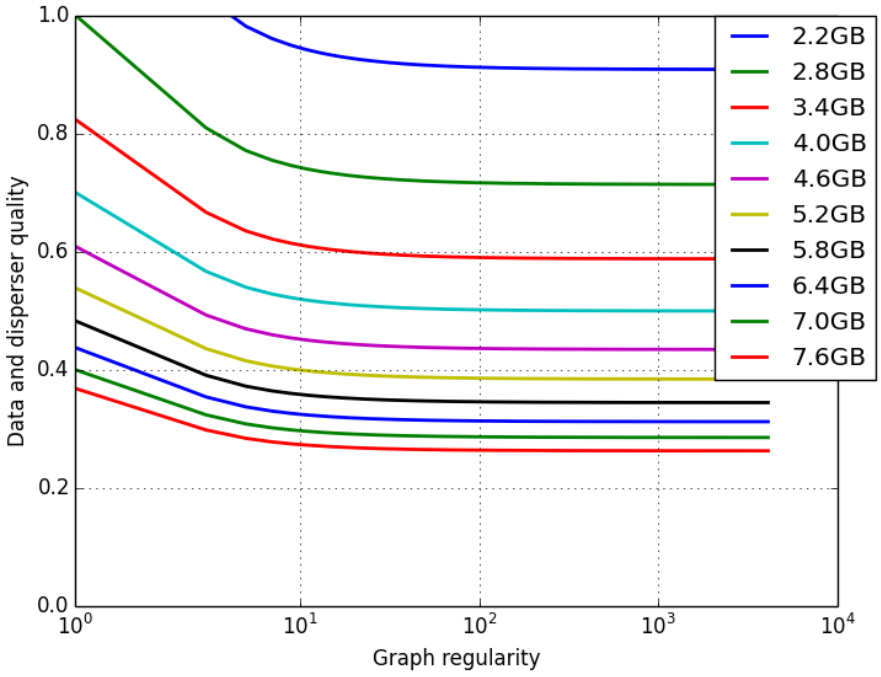


Figure 3.12.: Leakage 2 GB, 512-bit long output of a hash function

CHAPTER 4

KDF IN IDENTIFICATION AND SIGNATURE SCHEMES

In the previous parts of this work we emphasized that the main practical application of the presented primitive focuses on making BRM schemes usable on mobile devices. One of the fundamental cryptographic operation performed on such devices is authentication, which allows remote machines to establish a secure connection between each other. With no exaggeration one may claim that secure Internet browsing depends on effective identification schemes.

Identification is a must for such vulnerable applications like these which manage user's bank account or email, where any failure can cause irretrievable damages. In modern days, we identify ourselves almost every time we request access to any remote resources, even the simplest one: application that manages our emails on a smartphone identify every time it checks whether new messages arrived, social media applications identify and check whether new posts were published, etc.

In this part of the thesis, we present a BRM identification scheme depending on a key obtained by kdf on the fly from disk data, instead of a key permanently resident in device's memory. Furthermore, we present a simple yet effective signature scheme based on identification scheme and Fiat-Shamir paradigm. The approach described here is well known in the cryptographic community. However, we show it in a new setting and prove security for much more powerful adversaries equipped with access to a leakage oracle. Finally, we analyse its usefulness to the problem described in the thesis.

4.1. Identification in the Bounded Retrieval Model

As stated before, BRM comes with an inevitable use of huge keys. This obstacle forces us to use asymmetric cryptography primitives, where machines taking part in the communication do not need to store a separate, huge cryptographic key for every other party.

However, naïve use of asymmetric cryptography is not a straight forward solution, either. If both public and private keys are large, the communication in a bigger network is also merely possible. The problem comes from the fact that each computer in the network has to store a, potentially huge, public key of every other machine. This obstacle forces us to use asymmetric cryptography, but in a setting, where public keys are short compared to private ones. A huge private key is possessed by a single party only, while others have a short public key that can be used to verify identification requests or validate signatures. Note that only private keys are prone to leakage, since public ones are known.

Due to a huge size of a BRM key we cannot use the entirety of it during the computations, because of large computation overhead. Popular approach solves this problem by using some randomly picked parts of the key and building a session key from them. However, this usually means that the session key changes every time, what rises another problem.

Consider a BRM identification protocol. In the first session an identifying party, the prover, uses randomness r_0 on a BRM key K that gives a short session key K_0. The prover uses K_0 to prepare her challenge response Prove. For the next session randomness changes, it is r_1 now what gives key K_1 out of the same BRM key K. How can the verifier check that both K_0 and K_1 comes from the same entity, from the same BRM key K? Maybe there is another party that picked some other K_1' maliciously and claim it as a legitimate key obtained from K? Compare this to an identification done by using some public-key scheme like RSA. Here a pair of keys $(\mathtt{pk}, \mathtt{sk})$ remains the same during the whole life of the protocol. For every $\mathtt{pk}$ there is exactly one $\mathtt{sk}$ and vice versa. In the BRM setting we would like to have multiple secret keys $\mathtt{sk}_0, \mathtt{sk}_1, \ldots$ (possibly one for each identification) but still a single public key $\mathtt{pk}$.

The identification scheme defined in Definition 12 consists of subroutines $(\mathsf{ParamGen}, \mathsf{KeyGen}, \mathsf{Prove}, \mathsf{Verify})$. Here, we point out two algorithms for Prove that makes identification possible even for a constantly changing session key:

- First, algorithm $\mathsf{GetSK}(K; r)$, returns a session key K_r from the BRM key K and given randomness.
- Second, algorithm $\mathsf{ProveSK}(K, K_r)$, that on a session key K_r produces an argument π which can be submitted to the verifier to convince him that K_r is a legitimate session key obtained from K.

We would like these subroutines to have the following properties:

Completeness An honest prover is able to produce argument π that is acceptable by an honest verifier with overwhelming probability. That is,

- for every uniformly random r,
- session key $K_r \leftarrow \mathsf{GetSK}(K; r)$, and
- $\pi \leftarrow \mathsf{ProveSK}(K, K_r)$,

$\mathsf{Verify}(\mathtt{pk}; \pi; r)$ returns `Accept`.

Soundness We say that Verify procedure is sound if the probability that it returns `Accept` on some π' such that there is no valid session key $K_{r'}$ for which $\pi' \leftarrow \mathsf{ProveSK}(K, K_{r'})$ is negligible.

In this part of the thesis we will provide a scheme that from a BRM key K builds session key K' and proves that K' was obtained honestly. In other words, we show here a concrete example of functions GetSK and ProveSK.

4.2. Construction overview

Assume that we have obtained a BRM key K. The method how the key was obtained is not relevant now, it can be either a truly random key or a key output by the proposed kdf. In this part we will show how to extract a session key K' from K and propose a simple yet efficient function GetSK. Furthermore, we show a protocol ProveSK that run on some key K' identifies the user by arguing that K' was produced out of key K (with overwhelming probability).

Here we focus on identification functionality where a user (client) needs to assure another user (server) that she is legitimated to perform certain operations. However, means proposed here can be used in other purposes, like signature schemes, with little change.

In the proposed setting, the identifying client asks for a random r that determines which part of the key she is going to use. This random r can be either sent by the server, e.g. through secure channel, posted on a bulletin board, obtained from an external but trusted common source of randomness, or just computed by the client and the server jointly. Because of security reasons, the client cannot pick r herself. In such case the adversary who learnt some part of the client's key could just decide to build a session key using this leaked part. Thus, uncertainty which key parts will be picked is a must.

Recall that in our model, the adversary learns leakage before she tries to identify as a legitimate user. Hence, we can safely assume that the server publishes random r on some sort of publicly available and verifiable bulletin board. Even if the adversary learns r she cannot adjust her leakage to it. However, it may happen that picked r fits her well. On the other hand, we will show that such an event occurs with a small, bounded probability.

Function GetSK. For given r, the client builds session key K_r by calling $\mathsf{GetSK}(K; r)$. In our proposal GetSK function is really simple. Suppose that the BRM key is divided into ℓ disjoint parts $K[1], \ldots, K[\ell]$. Session key $\mathsf{GetSK}(K; r) = K_r$ is created by assigning to it $K[r \bmod \ell]$. Here, the client and the server need to keep track of r used as no key K_r can be used twice. Thus, if the server asks for creating K_r for some r that has been used previously, such a demand must be rejected. In the next part it will be explained why using the same key twice is not secure. We assume that since random values r are publicly known, the server sends a new, fresh r' every time.

Function ProveSK**.** Function ProveSK is based on the following idea: a legitimate user is able to show a sequence of transformations of the session key such that in the end these transformations result in a public key `pk`. Furthermore, no one without a knowledge of the session key can do the same (with non-negligible probability).

Let $\mathcal{T}$ be a binary tree that has leaves labeled by the consecutive blocks of key K, i.e. first leaf stores $K[1]$, second stores $K[2]$, etc. and stores `pk` as a label of the root. Function ProveSK will return a path from a leave to the root. To provide security of such a construction, we will use Merkle trees. This assures that the adversary who returns acceptable $\mathsf{ProveSK}(K'; r)$ has access to the key K with overwhelming probability.

4.3. Merkle tree

The ProveSK function is based on a construction called Merkle tree [Mer88] (cf. Figure 4.1). This structure is a binary tree with a hash function $h : \{0,1\}^{2n} \to \{0,1\}^n$ from a collision resistant hash function family (see Def. 31). Tree nodes have assigned values according to the following rule: every leaf a is labeled by some number $a.\texttt{label}$ in $\{0,1\}^n$. Since we do not store in nodes any other values, we identify value of a node with a value stored in its label. Thus, we will omit "`.label`" part further on and sometimes abuse notation writing that at node x^i_j we store value x^i_j. If a node c has two children a, b, then c gets value $h(a, b)$.

Definition 31 (Collision resistant hash function family, see [KL07])**.** Let $\mathcal{H}$ be a family of functions $h_K : \mathbf{KeySpace} \times \{0,1\}^{2n} \to \{0,1\}^n$. We say that $\mathcal{H}$ is collision resistant if for any PPT adversary $\mathcal{A}$

$$\Pr(K \leftarrow \mathbf{KeySpace}, (m, m') \leftarrow \mathcal{A}(K) : m \neq m' \wedge h_K(m) = h_K(m'))$$

is negligible.

For sake of brevity, we will write h instead of h_K, as if the function was not keyed. Similarly, we will write that h maps $\{0,1\}^{2n}$ to $\{0,1\}^n$ and omit set **KeySpace**. In our case we will use random oracle $\mathcal{H}$ as function h. The description of a Merkle tree can be formalized as:

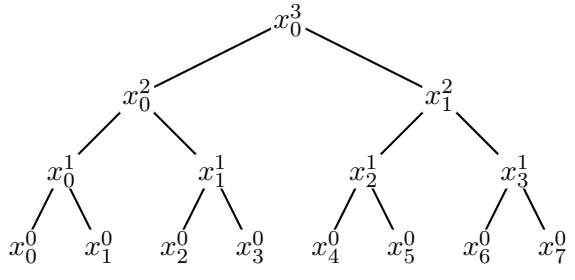
Definition 32 (Merkle tree for hash function h, [Mer88])**.** Let $\mathcal{T}(\ell)$ denote a full binary tree of ℓ leaves and height $\log \ell$, let x^0_i (for $i \in \{0, \ldots, \ell - 1\}$) be leaves in $\mathcal{T}(\ell)$ and $x^0_i \in \{0,1\}^n$ be a value stored in x^0_i. For every node $x^j_i \in \mathcal{T}(\ell)$ that is not a leaf we denote by x^{j-1}_{2i} and x^{j-1}_{2i+1} the direct ancestors of node x^j_i. We call such tree $\mathcal{T}(\ell)$ Merkle tree for function $h : \{0,1\}^{2n} \to \{0,1\}^n$, if holds:

$$x^j_i = h\left(x^{j-1}_{2i}, x^{j-1}_{2i+1}\right), \text{ for } j > 0 \text{ and some } x^0_0, x^0_1, \ldots, x^0_{\ell-1} \ .$$

Remark. Note that we defined Merkle tree regarding to a collision-resistant hash function h. However, to show some properties of this primitive we will substitute h by a random

oracle $\mathcal{H}$.

An illustrative example of a Merkle tree is given on Figure 4.1. On Figure 4.2 we provide an exemplary implementation of such a structure. The idea of the implementation is simple and goes as follows. Let ℓ be the number of leaves and a power of 2. We represent the tree as a one-dimensional array of length $2\ell - 1$. First ℓ entries are occupied by labels stored in the leaves while the others remain empty. Then going from left to right we pair given values and store the hash of the pair in the first empty cell.



$$x_0^3 = h(x_0^2, x_1^2)$$

$$x_0^2 = h(x_0^1, x_1^1), \quad x_1^2 = h(x_2^1, x_3^1)$$

$$x_0^1 = h(x_0^0, x_1^0), \quad x_1^1 = h(x_2^0, x_3^0)$$
$$x_2^1 = h(x_4^0, x_5^0), \quad x_3^1 = h(x_6^0, x_7^0)$$

Figure 4.1.: An exemplary full Merkle tree

Implementation of makeTree.

Input: One-dimensional array Q of length $2\ell - 1$ and ℓ values $x_0, \ldots, x_{\ell-1} \in \{0,1\}^n$ to be stored at leaves of a tree; hash function $h : \{0,1\}^{2n} \to \{0,1\}^n$.
Output: Merkle tree $\mathcal{T}(\ell)$ represented by a one-dimensional array.

Algorithm 2: makeTree$(Q, x_0, \ldots, x_{\ell-1})$

```
1  length ← ℓ
2  length.last ← ℓ
3  length.last.prev ← 0
4  while length ≥ 2 do
5  |  for i = 0, ..., length/2 − 1 do
6  |  |  Q[i + length.last] ←
   |  |    h(Q[length.last.prev + 2i], Q[length.last.prev + 2i + 1])
7  |  end
8  |  length.last.prev ← length.last
9  |  length ← length/2
10 |  length.last ← length.last + length
11 end
```

Figure 4.2.: Building a Merkle tree from data blocks

Merkle tree is a construction widely used in cryptography, since it straightforwardly gives a hash function able to obtain an input of arbitrary length, say ℓn, given only hash

function $h : \{0,1\}^{2n} \to \{0,1\}^n$ in $\log \ell$ steps. Furthermore, it gives a simple identification protocols as described in Section 4.4.

In the next definition we will describe an identification path for a leaf i in a Merkle tree. This definition will be crucial for understanding properties of ProveSK algorithm. Informally, an identification path is a sequence of all nodes laying on the path from the leaf to the root along with all their sibling nodes.

Definition 33 (Identification path in a Merkle tree, see [Mer79])**.** Let $\mathcal{T}(\ell)$ be a Merkle tree of height $\log \ell$ with leaves $x_0, \ldots, x_{\ell-1} \in \{0,1\}^n$ and hash function $h : \{0,1\}^{2n} \to \{0,1\}^n$, the sequence $\mathsf{p} = (x_0^0, x_1^0, x_0^0, x_1^1, \ldots, x_0^{\log \ell - 1}, x_1^{\log \ell - 1}, x) \in (\{0,1\}^n)^{2\log \ell + 1}$ is called an *identification path from leaf* x_i if

- $x_0^0 = x_i$ if i even and $x_1^0 = x_i$ otherwise;
- let $\vec{b} = b_0, b_1, \ldots, b_{\log \ell - 1}$ such that $i = b_0 + 2 \cdot b_1 + \ldots + 2^{\log \ell - 1} b_{\log \ell - 1}$, where $b_j \in \{0,1\}$ then the following holds

$$\text{for } b_j = 0 \qquad x_0^j = h\left(x_0^{j-1}, x_1^{j-1}\right) \ ; \tag{4.1}$$

$$\text{for } b_j = 1 \qquad x_1^j = h\left(x_0^{j-1}, x_1^{j-1}\right) \ ; \tag{4.2}$$

  where $0 < j < \log \ell$; and
- $x = h\left(x_0^{\log \ell - 1}, x_1^{\log \ell - 1}\right)$.

  Node $v \in \mathsf{p}$ such that Eq. (4.1) or (4.2) holds is called a *path node*, otherwise we call it a *sibling node*. The set of all path nodes is denoted by **Pth** and the set of all siblings node is denoted by **Sbl**.

Figure 4.3 shows an exemplary implementation of an algorithm that returns the identification path for the required node. Since we include in the identification path also siblings of nodes we can verify it easily, see Figure 4.5.

4.4. Identification on a Merkle tree

Identification by using Merkle tree is a well-known technique. It allows the prover to identify using logarithmically long (in numbers of leaves) proof, while the only element needed to be known by the verifier is the value at the root of the tree. Despite popularity of the scheme, all security proofs were done according to a model where leakage does not occur and under assumption that values stored at leaves are uniformly random.

Here, we modify the identification protocol to make sure that even the adversary who learnt some information about the tree cannot break the soundness of the identification. More precisely, we will prove security of the identification scheme by showing that no

Implementation of getPath

Input: Full Merkle tree $\mathcal{T}(\ell)$, with ℓ leaves $x_0^0, \ldots, x_{\ell-1}^0$ and root c, each of which labeled by a number from $\{0,1\}^n$, leaf index k written binary as $b_0 b_1 \ldots b_{\log \ell}$.

Output: Path $\mathsf{p} = \left(y_0^0, y_1^0, y_0^1, y_1^1, \ldots, y_0^{\log \ell - 1}, y_1^{\log \ell - 1}, y\right) \in (\{0,1\}^n)^{2 \log \ell + 1}$, such that for $i \in \{0, \ldots, \log \ell - 1\}$

- $(y_0^i, y_1^i) = \begin{cases} h\left(x^i_{\lfloor \frac{k}{2^i} \rfloor}, x^i_{\lfloor \frac{k}{2^i} \rfloor + 1}\right), \text{ if } b_i = 0, \\ h\left(x^i_{\lfloor \frac{k}{2^i} \rfloor - 1}, x^i_{\lfloor \frac{k}{2^i} \rfloor}\right), \text{ if } b_i = 1 \end{cases}$
- $y = x_0^{\log \ell}$

Figure 4.3.: Algorithm getPath obtains a path from a leaf of index k up to the root c of tree $\mathcal{T}(\ell)$.

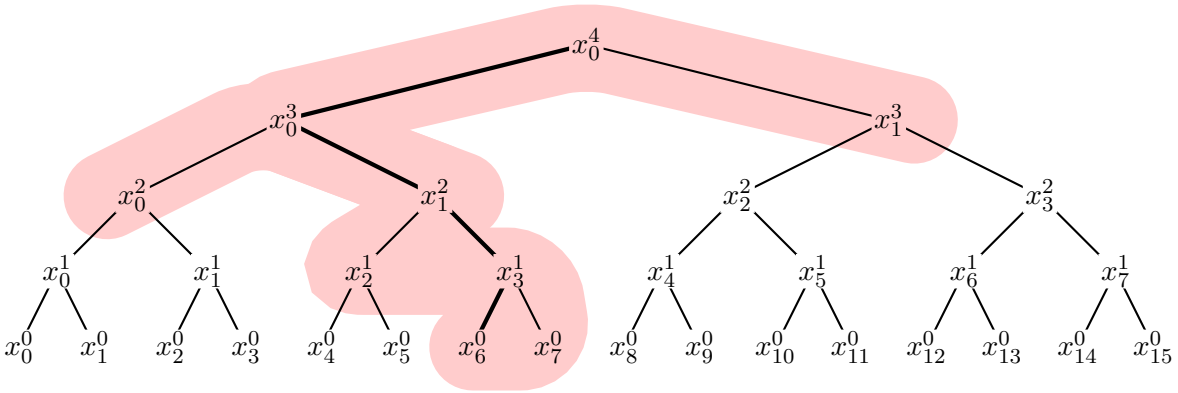


Figure 4.4.: An exemplary identification path on a Merkle tree. The highlighting denotes nodes on the identification path.

adversary $\mathcal{A}$ equipped with $q(n)$ queries to the random oracle and leakage λ has non-negligible probability of making verifier $\mathcal{V}$ accept $\mathcal{A}$ on behalf of a legitimate prover $\mathcal{P}$. This identification scheme allows the verifier to learn some small, yet non-negligible, part of the key. A direct consequence of this fact is that the prover cannot use this scheme as long as she wish and at some moment both private and public keys need to be refreshed. This problem is addressed later. For now, we assume that the prover is able to identify up to $\zeta(n)$ times.

The identification protocol is simple and goes as follows. We assume that the prover stores her cryptographic key (which can be computed from her private data) divided into ℓ blocks along with Merkle tree $\mathcal{T}(\ell)$ built on top of the key (i.e. with blocks of the key assigned to the tree leaves values). The verifier gets as public key pk the value stored in the root of tree $\mathcal{T}$. Both parties hold states, i.e. they remember what randomness they used in the past, what messages they received etc.

The identification starts with the verifier sending $k(n)$ different random numbers $r_i \in$

Implementation of checkPath.

Input: $\mathsf{p} = \left(y_0^0, y_1^0, \ldots, y_0^{\log \ell - 1}, y_1^{\log \ell - 1}, y\right) \in (\{0,1\}^n)^{2\log \ell + 1}$, an index k and a value c of the root of $\mathcal{T}(\ell)$.

Output: `TRUE` iff p is a correct path from k^{th} leaf up to the root c and `FALSE` otherwise.

Algorithm 3: checkPath(p, k, c)

Let $b_{\log \ell} \ldots b_1 b_0$ be a binary representation of k. Then:
for $i = 1, \ldots, \log \ell - 1$ **do**
  if $y_0^i \neq h\left(y_0^{i-1}, y_1^{i-1}\right)$, *for* $b_i = 0$ **then**
    Return `FALSE`
  end
  if $y_1^i \neq h\left(y_0^{i-1}, y_1^{i-1}\right)$, *for* $b_i = 1$ **then**
    Return `FALSE`
  end
end
if $y \neq h\left(y_0^{\log \ell - 1}, y_1^{\log \ell - 1}\right)$ **then**
  Return `FALSE`
end
if $y \neq c$ **then**
  Return `FALSE`
end
Return `TRUE`

Figure 4.5.: Algorithm checkPath checks whether the given path p is a correct path from the leaf of index k up to the root c.

$\{0, \ldots, \ell - 1\}$ that were not used before. On every r_i the prover responds with a path in $\mathcal{T}(\ell)$ that connects r_i-th leaf with the root along with siblings of every node on the path. See Definition 33 and Figure 4.3 for an algorithm that returns required path. The last step in the protocol is done by the verifier who checks correctness of the given paths (see Figure 4.5) and outputs `Accept` if all paths were computed correctly and `Reject` if this is not the case. The detailed description of the identification protocol appears on Figure 4.6.

We claim that such identification scheme is secure even if the adversary is allowed to learn some part of the secret key. The proof idea goes as follows. The key is divided into blocks and, due to the construction of the key, these blocks are independent of each other. Recall, this property holds for a uniformly random key and a key produced by Disperse alike. The adversary submits leakage queries of overall length not exceeding λ. Such queries reduces min-entropy of the key by at most λ. We conclude that some of key blocks have now entropy reduced to some small value, while entropy of other blocks remains (mostly) unaffected. During the identification, the verifier asks for paths to $k(n)$ leaves. We argue, that if among these $k(n)$ paths there is at least one leaf with high min-entropy, then the adversary wins only with negligible probability. Then we show

Merkle tree identification protocol

Setup phase:

ParamGen :

Both parties, a prover $\mathcal{P}$ and a verifier $\mathcal{V}$, set $\mathbf{S} \leftarrow \emptyset$ and keep independent copies $\mathbf{S}_{\mathcal{V}}$ and $\mathbf{S}_{\mathcal{P}}$ of $\mathbf{S}$.

KeyGen: Prover $\mathcal{P}$ picks $\mathtt{sk} = x_0, \ldots, x_\ell$ randomly from $\{0,1\}^{n\ell}$, computes $\mathcal{T}$ – a Merkle tree with $x_0, \ldots, x_{\ell-1} \in \{0,1\}^{n\ell}$ as values at leaves. Then, she computes $\mathtt{pk}$, the value of the root of tree $\mathcal{T}$ and sends it to verifier $\mathcal{V}$.

Challenge:

Verifier $\mathcal{V}$ picks at random vector $\vec{v} = (v[1], v[2], \ldots, v[k(n)]) \in \{0, \ldots, \ell-1\}^{k(n)}$, such that for $i = 1, \ldots, k(n)$,

- $v[i] \leftarrow \{0, \ldots, \ell-1\} \setminus \mathbf{S}_{\mathcal{V}}$;
- for $j \neq i$,
  - $v[i] \neq v[j]$,
  - $v[i] \neq v[j] + (-1)^{v[j]}$

and sends the chosen vector to $\mathcal{P}$;
$\mathbf{S}_{\mathcal{V}} \leftarrow \mathbf{S}_{\mathcal{V}} \cup \left\{v[i], v[i] + (-1)^{v[i]}\right\}_{i=1,\ldots,k(n)}$ (where $i + (-1)^i$ is an index of i's sibling leaf).

Prove : For $i = 1, \ldots, k(n)$, prover $\mathcal{P}$

- checks whether $v[i] \in \mathbf{S}_{\mathcal{P}}$ and aborts if this is the case, otherwise sets $\mathbf{S}_{\mathcal{P}} \leftarrow \mathbf{S}_{\mathcal{P}} \cup \{v[i], v[i] + (-1)^{v[i]}\}_{i=1,\ldots,k(n)}$.
- computes values lying on the path from $\mathtt{sk}_{v[i]}$ up to $\mathtt{pk}$ along with values at sibling nodes, i.e. runs

$$\mathsf{p}_{v[i]} \leftarrow \mathsf{getPath}(\mathcal{T}, v[i])$$

and sends $\mathsf{p}_{v[i]}$ to $\mathcal{V}$.

Verification:

Verify : Verifier $\mathcal{V}$ checks whether for each i in $1, \ldots, k(n)$ on obtained values $\mathsf{p}_{v[i]}$ holds

$$\mathsf{checkPath}(\mathsf{p}_{v[i]}, v[i], \mathtt{pk})$$

If this is the case, the verifier outputs `Accept`, if not, `Reject`.

Figure 4.6.: Merkle tree identification protocol

that with overwhelming probability among $k(n)$ random leaves there is a leaf of high min-entropy.

Theorem 16 (Identification security)**.** *For any $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}^{K,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}$ probability that $\mathcal{A}$ wins in a $\zeta(n)$-bounded leakage-resilient identification scheme security game described on Figure 2.7 for identification on Merkle tree (see Figure 4.6), for key K picked uniformly at random from $\{0,1\}^{n\ell}$, $\zeta(n) < \ell/2k(n)$ and $\lambda \leqslant \frac{1}{c} \cdot (\ell - 2k(n)\zeta(n))(n - \log(q(n)r(n)))$, is not greater than*

$$\left(\frac{1}{c}\right)^{k(n)} + \frac{1}{r(n)-1} \ ,$$

for any $r(n) > 1, c > 1$.

Before we proceed with the proof, we present some intuitions behind the theorem. The bound on the number of executions $\zeta(n)$ comes from the number of available paths in a Merkle tree with ℓ leaves (each leaf is a block of a key). Every identification reduces the number of available paths by $2k(n)$. More precisely, in every identification execution, the verifier asks for $k(n)$ paths and because of path verification algorithm, sibling leaves share the same identification path, thus every path query excludes two leaves.

We say that a block has *small* min-entropy if the random variable describing values the node can take has min-entropy smaller than $\log(q(n)r(n))$ after the adversary's leakage queries. We say that a block has *high* min-entropy if it does not have small min-entropy. We assume that the adversary knows all blocks with small min-entropy. That is, every block has min-entropy either at least $\log(q(n)r(n))$ or 0. Since the min-entropy of each block before the leakage is n, adversary $\mathcal{A}$ has to use at least $n - \log(q(n)r(n))$ of her leakage to make a block have small min-entropy. We show later that the probability of the adversary guessing correctly value of a block of high min-entropy is negligible.

The adversary can make her leakage queries before any of $\zeta(n)$ runs of the identification protocol. In such a protocol, the verifier requires the prover to provide him a number of identification paths from the key blocks up to the root of the tree. In some round of identification the adversary decides to end *Test stage* and go to *Impersonation stage*, see Figure 2.7.

The bound on the leakage comes from the following. We allow the adversary to leak as much as she wants, providing that the number of blocks with high min-entropy remains big enough to have overwhelming probability that the verifier picks such a block in her identification challenge.

Let us comment on the probability of breaking security of the scheme. The first ingredient $c^{-k(n)}$ is negligible for $k(n)$, if $c > 1$. The second; although the proof works for any $r(n) > 1$, $1/r(n)$ has to be negligible in n for α to be negligible also. Below we continue with a proof of Thm 16.

Proof of Thm. 16. Let $\mathcal{T}_0 = \left(x^0_1, \ldots, x^0_\ell, x^1_0, \ldots, x^1_{\ell/2}, \ldots, x^{\log \ell}_0\right)$ denote the Merkle tree build upon the key $K = x^0_1, \ldots, x^0_\ell$ and $\mathcal{T}_1$ denote $\mathcal{T}_0$ with leaves truncated, i.e. $\mathcal{T}_1 = \left(x^1_0, \ldots, x^1_{\ell/2}, \ldots, x^{\log \ell}_0\right)$. We will prove a stronger theorem by supplying the adversary

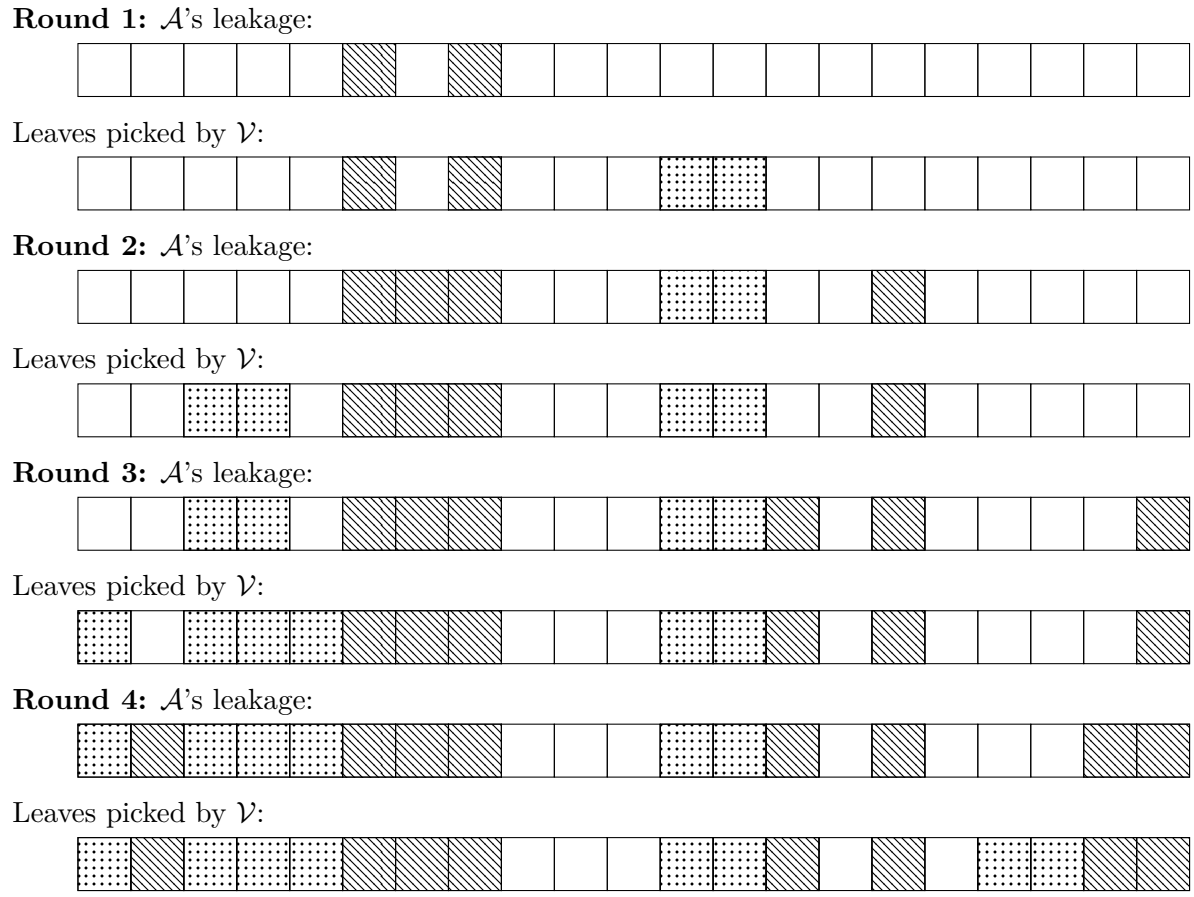


Figure 4.7.: Exemplary execution of a few rounds of the identification protocol. Before each round adversary $\mathcal{A}$ picks leaves she wants to learn (blocks filled with slanted lines), then verifier $\mathcal{V}$ picks (independently) leaves for the identification (blocks filled with dots). We show here very lucky adversary whose leakage queries have empty intersection with challenges sent by the verifier.

with additional knowledge. More precisely, the values from the truncated tree $\mathcal{T}_1$. That is, the adversary is given all possible paths from the bottom to the top of a tree $\mathcal{T}_0$ except the values at the leaves.

Suppose that adversary $\mathcal{A}$ plays a $\zeta(n)$-security game of the identification scheme. The identification procedure is executed up to $\zeta(n)$ times. At some moment adversary $\mathcal{A}$ decides to identify. (goes from the *Test stage* to the *Impersonation stage*). We say that $\mathcal{A}$ wins if she successfully identify to verifier $\mathcal{V}$. We consider two complementary events:

- E – in the $k(n)$ paths requested by the verifier there is at least one path pointing to a leaf with min-entropy at least $\log(q(n)r(n))$. Recall, we say that a leaf v has small min-entropy if $\mathrm{H}_\infty(V) \leqslant \log(q(n)r(n))$, for V a random variable describing possible values of v. As mentioned above, we assume that all leaves with small min-entropy are known to the adversary. That is, leaves has either min-entropy at least

$\log(q(n)r(n))$ or 0.

- E' – is an event complementary to E. That is, E' occurs when verifier $\mathcal{V}$ requests paths to leaves that are all known to adversary $\mathcal{A}$.

Now, the probability of winning (in Figure 2.7) can be expressed as

$$\Pr(\mathcal{A} \text{ wins}) = \Pr(\mathcal{A} \text{ wins} \mid E)\Pr(E) + \Pr(\mathcal{A} \text{ wins} \mid E')\Pr(E') \ .$$

The proof will show that both $\Pr(\mathcal{A} \text{ wins} \mid E)$ and $\Pr(E')$ are bounded by $1/(r(n)-1)$ and $c^{-k(n)}$ respectively.

We start by showing that $\Pr(\mathcal{A} \text{ wins} \mid E) \leqslant 1/(r(n)-1)$. Consider the verifier's request on some path ended in a leaf v_l. For sake of simplicity we assume that it is the left sibling, we will denote the right sibling by v_r. Denote by v the parent of v_l and v_r. Since the adversary is equipped with $\mathcal{T}_0$, the only thing she has to compute before she can answer the verifier's request is a value on v_l, v_r such that $\mathcal{H}(v_l, v_r) = v$. Without losing the generality, assume that $\mathrm{H}_\infty(V_l) \geqslant \log(q(n)r(n))$ and v_r is known to $\mathcal{A}$.

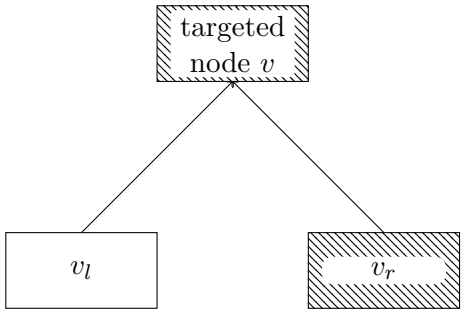


Figure 4.8.: Adversary $\mathcal{A}$ needs to find such a value of v_l, v_r that $\mathcal{H}(v_l, v_r) = v$. By the assumption, v_r and v are known to $\mathcal{A}$.

Because of the random oracle properties, the adversary equipped with a single query to oracle $\mathcal{H}$ has the probability of guessing value of v_l bounded by $1/(q(n)r(n))$. Since the adversary is equipped with $q(n)$ queries, the probability of a correct guess in the i-th query is $1/(q(n)r(n)-i)$. Finally, the probability that $\mathcal{A}$ guesses the value of v after $q(n)$-th query is $1/q(n)(r(n)-1)$. Since the adversary wins if she makes a correct guess in at least one query and by the union bound on probability we have

$$\Pr(\mathcal{A} \text{ guesses the value of } v_l) \leqslant \underbrace{\frac{1}{q(n)r(n)} + \frac{1}{q(n)r(n)-1} \cdots + \frac{1}{q(n)(r(n)-1)}}_{q(n) \text{ times}} \leqslant \frac{1}{r(n)-1} \ .$$

The next step is to show that probability that all leaves pointed by the chosen $k(n)$ paths have min-entropy smaller than $\log q(n)r(n)$ is bounded by $c^{-k(n)}$. Denote by $\mathbf{T}$ a set of all leaves with min-entropy not greater than $\log(q(n)r(n))$. Because values of all leaves are taken randomly from $\{0,1\}^n$ and amount of leakage does not exceed λ,

therefore $|\mathbf{T}| \leqslant \lambda/(n - \log(q(n)r(n)))$. For round i denote by $\mathbf{T}_i$ all nodes known to the adversary after she makes her leakage queries. Recall, that the adversary cannot make leakage queries during the identification protocol execution, but can make them in between executions. Suppose, the adversary leaks λ_i bits of information right before the i-th identification execution. It holds that $|\mathbf{T}_i| \leqslant (\lambda_1 + \ldots + \lambda_i)/(n - \log(q(n)r(n)))$.

The adversary picks which round she will try to identify. Say she decided on round i. The probability that the adversary wins in round i is bounded by

$$\left(\frac{|\mathbf{T}_i|}{\ell - 2k(n)i}\right)^{k(n)} \quad , \tag{4.3}$$

which holds since in round i (a) there are at most $|\mathbf{T}_i|$ nodes of small min-entropy; (b) in the beginning of that round there is exactly $\ell - 2k(n)(i-1)$ nodes left; (c) the verifier picks (independently) $k(n)$ nodes; (d) the probability that the verifier pick the j-th node in the i-th round from the set of nodes known to the adversary is at most

$$\frac{|\mathbf{T}_i|}{\ell - 2k(n)(i-1) - 2j + 2} \leqslant \frac{|\mathbf{T}|}{\ell - 2k(n)i} \quad .$$

For $\lambda \leqslant \frac{1}{c}(\ell - 2k(n)\zeta(n))(n - \log(q(n)r(n)))$ Eq. (4.3) is bounded by $c^{-k(n)}$ for some $c > 1$. Putting both parts together, we obtain the requested bound

$$\left(\frac{1}{c}\right)^{k(n)} + \frac{1}{r(n) - 1} \quad .$$

That finishes the proof. □

The theorem can be illustrated by the following corollary:

Corollary 17. *For $r(n) \geqslant 2^{256}, c = 2$ and $k(n) = 256$, any $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}^{K,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}$, probability that $\mathcal{A}$ wins in a $\zeta(n)$-bounded leakage-resilient identification scheme security game described on Figure 2.7 for identification on Merkle tree (see Figure 4.6), for key K picked uniformly at random from $\{0,1\}^{n\ell}$, $\zeta(n) < \ell/512$ and $\lambda \leqslant \frac{1}{2} \cdot (\ell - 512\zeta(n))(n - \log q(n) - 256)$, is not greater than 2^{-128}.*

Note, the proof above assumed that the key K is uniformly random. However, we could substitute K by a key prepared by a secure kdf with only negligible loss in security. We conclude the discussion with the following corollary.

Corollary 18. *For any $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}^{D,\mathcal{H}},\mathcal{H}}_{\lambda,q(n)}$ probability that $\mathcal{A}$ wins in a $\zeta(n)$-bounded leakage-resilient identification scheme security game described on Figure 2.7 for an identification on a Merkle tree (see Figure 4.6), for key of length $\{0,1\}^{n\ell}$ produced by a $(\Lambda(\lambda), q(n), \varepsilon(n))$-secure kdf, $\zeta(n) < \ell/2k(n)$ and $\lambda \leqslant \frac{1}{c} \cdot (\ell - 2k(n)\zeta(n))(n - \log(q(n)r(n)))$ is not greater than*

$$\left(\frac{1}{c}\right)^{k(n)} + \frac{1}{r(n) - 1} + \varepsilon(n) \quad ,$$

for any $r(n) > 1$, $c > 1$.

Proof. Bound on probability α comes directly from Theorem 16 and a security gap between running the protocol on a BRM key generated uniformly at random and a key derived by the kdf. □

If we use described in Chapter 3, $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}$ as kdf, we get the following parameters.

Corollary 19. *For any* $\mathcal{A} \in \mathbf{TM}^{\mathcal{O}^{D,\mathcal{H}},\mathcal{H}}_{\lambda-\ell^e(\log q(n)+n),2^{o(\ell^{1-e})}}$ *probability that* $\mathcal{A}$ *wins in a* $\zeta(n)$*-bounded leakage-resilient identification scheme security game described on Figure 2.7 for an identification on a Merkle tree (see Figure 4.6), for key of length* $\{0,1\}^{n\ell}$ *produced by a* $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}$ *on data* D *such that* $\mathrm{H}_\infty(D) \geqslant pn\ell$ *and* $\mathcal{G}_\sigma$ *a* d*-regular* (ℓ^e,η)*-right disperser is not greater than*

$$\left(\frac{1}{c}\right)^{k(n)} + \frac{1}{r(n)-1} + 2^{-n((p-\eta)\ell)+\lambda+\ell^e(\log q(n)+\log \ell)} \ ,$$

for any $r(n) > 1$, $c > 1$.

4.5. Non-interactive identification based on a Merkle-tree

The aforementioned scheme can be easily made non-interactive by using the Fiat-Shamir paradigm. A non-interactive scheme allows prover $\mathcal{P}$ to send a single message, which fully identifies her without any interaction with the verifier.

Similarly to the construction described in Section 4.4, a non-interactive scheme consists of three messages. However, instead of a challenge picked by the verifier, the second message comes from querying the random oracle on some one-time, publicly known randomness r picked by the prover, see Figure 2.3. The detailed construction of a scheme can be found on Figure 4.9.

Before we proceed, we assume that there exists a random oracle $\mathcal{H}^{[k(n)]}$, which on query r produces a binary string $\mathcal{H}^{[k(n)]}(r)$ of length ℓ, such that the Hamming weight of a binary representation of $\mathcal{H}^{[k(n)]}$ is exactly $k(n)$. Exemplary construction of such random oracle can be given according to [HJK+16].

4.6. Efficiency

In this section we would like to give some estimation on cost of operations that a prover has to perform to identify to a verifier. We will also show bounds on the number of bits that has to be sent from $\mathcal{P}$ to $\mathcal{V}$.

For given parameter $k(n)$, the identifying prover has to reply on $k(n)$ verifier's queries. That is, she has to compute $k(n)$ blocks of the key, each of which requires to query a random oracle on d blocks of data, for d regularity of the disperser. With high probability, some of these blocks will overlap. However, the upper bound for the number of accessed

Non-interactive Merkle tree identification protocol.

Setup phase:

ParamGen : Prover $\mathcal{P}$: $\mathcal{T}$ – Merkle tree on leaf-values $x_1, \ldots, x_\ell$. Both parties set $\mathbf{S} \leftarrow \emptyset$ and keep independent copies $\mathbf{S}_\mathcal{V}$ and $\mathbf{S}_\mathcal{P}$ of $\mathbf{S}$. Set $\mathbf{R}$ denotes all random values that has been used so far, protocol begins with $\mathbf{R} \leftarrow \emptyset$.

KeyGen: Prover $\mathcal{P}$ computes $\mathtt{pk}$, the value of the root of tree $\mathcal{T}$ and sends it to verifier $\mathcal{V}$.

Challenge:

Prove: Prover $\mathcal{P}$

- takes at random $r \leftarrow \mathbf{Dom}(\mathcal{H}^{[k(n)]}) \setminus \mathbf{R}$;
- sets $\mathbf{R} \leftarrow \mathbf{R} \cup \{r\}$;
- query oracle $\mathcal{H}^{[k(n)]}$ on r obtaining $\mathcal{H}^{[k(n)]}(r)$, let $h_0 h_1 \ldots h_{\ell-1}$ be a binary representation of $\mathcal{H}^{[k(n)]}(r)$, set $\mathbf{K} \leftarrow \emptyset$;
- if for all h_i, $i = 0, \ldots, \ell - 1$, where $h_i = 1$, holds that $i \notin \mathbf{S}_\mathcal{P}$, then
  - for $i = 0, \ldots, \ell - 1$, if $h_i = 1$, set $\mathbf{K} \leftarrow \mathbf{K} \cup \{i\}$ and $\mathbf{S}_\mathcal{P} \leftarrow \mathbf{S}_\mathcal{P} \cup \{i\}$;
- else, if there is some i such that $h_i = 1$ and $i \in \mathbf{S}_\mathcal{P}$, prover $\mathcal{P}$ restarts and chooses another r.
- for every $i \in \mathbf{K}$ computes a path p_i from root to i^{th} leaf, i.e. runs $\mathsf{p}_i \leftarrow \mathsf{getPath}(\mathcal{T}, i)$;
- sends $\mathcal{V}$ a triple $\left(r, \mathcal{H}^{[k(n)]}(r), (\mathsf{p}_i)_{i \in \mathbf{K}}\right)$.

Verification:

Verify : On triple $\left(r, \mathcal{H}^{[k(n)]}(r'), (\mathsf{p}_i)_{i \in \mathbf{K}}\right)$ verifier $\mathcal{V}$ checks

- whether $r' \in \mathbf{R}_\mathcal{V}$ or $\mathcal{H}^{[k(n)]}(r') \neq \mathcal{H}^{[k(n)]}(r)$ if this happens $\mathcal{V}$ outputs `Reject`.
- For every $i \in \mathbf{K}$, $\mathcal{V}$ check whether:
  - $i \notin \mathbf{S}_\mathcal{V}$, $\mathbf{S}_\mathcal{V} \leftarrow \mathbf{S}_\mathcal{V} \cup \{i\}$;
  - $\mathsf{checkPath}(\mathsf{p}_i, r', \mathtt{pk})$.

If this is a case, the verifier outputs `Accept`, if not, `Reject`.

Figure 4.9.: Exemplary non-interactive Merkle tree identification protocol

blocks of disk data is $k(n) \cdot d$. Since every block is n bit long, it gives $k(n) \cdot d \cdot n$ bits that have to be read.

On the other hand, recall Theorem 15. According to it, the expected value of the number of data blocks that are read to get $k(n)$ key blocks from a disperser of regularity d is

$$\ell \left(1 - \left(1 - \frac{d}{\ell}\right)^{k(n)}\right) \ .$$

Identification requires $\mathcal{P}$ to send identification paths that correspond to the picked $k(n)$ key elements. Assuming that the key is built from ℓ blocks, a single path from the block to the tree root has length roughly $\log \ell$. Every node contains a value from $\{0,1\}^n$, thus the prover has to transmit $2 \cdot k(n) \cdot \log \ell \cdot n$ bits, where the factor 2 comes from the fact that for every node on the path, we also send its sibling.

Analysing Corollary 18, we can estimate the number above as follows: We pick a security parameter n equal 512 and $k(n) = 256$, for disk data of length 4 GB, we have $\ell \approx 8,4 \cdot 10^3$ and $\log \ell = 23$. Altogether, the number of bits send in a single identification query is roughly $2 \cdot 256 \cdot 23 \cdot 512 = 6\ 029\ 312$.

4.7. Signature scheme from kdf and a Merkle tree

A signature scheme based on a Merkle tree works analogously to a non-interactive identification scheme. The only difference is an additional randomizer r, that works twofold: it prevents the scheme from repetition attacks and allows to sign the same message twice. Thus, for σ being a signature we get $\sigma \leftarrow \Big((m,r), \mathcal{H}^{[k(n)]}(m,r), \mathsf{Prove}(\mathtt{sigk}; (m,r), \mathcal{H}^{[k(n)]}(m,r))\Big)$ for message m and randomizer r. After i identifications/signatures $2k(n)i$ paths are known, thus we need to assure that the paths pointed by $\mathcal{H}^{[k(n)]}(m,r)$ have not been used before. We denote such an event by $E(m,r)$. If this is not a case, we pick another randomizer r' and check paths for $\mathcal{H}^{[k(n)]}(m;r')$. New randomizers are checked until a value r', such that $E(m,r')$ occurs. Since in the single trial the probability of $E(m,r')$ is at least $\frac{\ell - 2k(n)i}{\ell}$, the expected number of trials before all picked paths were not used before is $\frac{2\ell}{\ell - 2k(n)i}$. That is, the probability that no suitable randomizer r is found in polynomial time is negligible. thus the probability that trying different randomness does not end in polynomial time is negligible. The security of the scheme is due to Corollary 4.

4.8. Public key updates

The scheme, as presented above, has a minor utility drawback. That is, it cannot be used with an unchanged key forever. After a few thousands of executions (depending on the parameters), when the number of identifications performed exceeds $\zeta(n)$, the scheme is no longer secure and a new key is necessary.

Another case which demands key refreshing comes from the nature of underlying data

D. As mentioned in the introduction, this research was inspired by the idea of using meaningful private data instead of a huge random sequences of bits to generate secure cryptographic keys. Since we do not want to limit functionality of these data, we have to allow their owner to deal with them as usual, i.e. add, remove, replace or modify them. However, properties of the proposed Disperse procedure makes these operations (despite addition) impossible. Every change in the data makes Disperse produce a different secret key, which yields a different public key.

Thus, to provide reliability and robustness, a protocol that allows the scheme to operate uninterruptedly is a must. Here we present a protocol that allows the user to update her public key when the old one was used too many times or it was changed due to changes in the underlying disk data. We show that the cost of updating the scheme is small. More precisely, it requires only a single authenticated message consisted of the new public key

The key updating procedure consists of two subroutines. Firstly, the new public key is computed. Secondly, it is *legalized*, that is, the user who updates the key has to prove that she is legitimized to do so, she is a legitimate user.

Below we present aforementioned two subroutines. The key updating procedure is presented on Figure 4.10. It needs a little modified Disperse algorithm, since the algorithm presented in Figure 3.2 does not include enumerator $\mathtt{ch} \in \{0,1\}^{\log \ell}$ that shows how many times the public key has been changed. For data $D_{\mathtt{ch}}$, graph $\mathcal{G}_\sigma$, and oracle $\mathcal{H}$ we denote the modified algorithm by $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D_{\mathtt{ch}}, \mathtt{ch})$ and the obtained key $D'_{\mathtt{ch}}$. The algorithm has been presented in Figure 4.11.

Obtaining a new public key As shown on Figure 3.2, secret key $\mathtt{sk} = D'_1, \ldots, D'_\ell$ is derived by $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D)$ procedure. The consecutive D'_i are obtained as follows:

$$D'_i \leftarrow \mathcal{H}\left(i, D_{\sigma(i,1)}, \ldots, D_{\sigma(i,d)}\right) \ ,$$

for D_i being disk data and $i = 0, \ldots, \ell - 1$. Then the public key $\mathtt{pk}$ is obtained as a root of a Merkle tree with values assigned to leaves equal $D'_1, \ldots, D'_\ell$.

We change this procedure and introduce a new enumerator $\mathtt{ch}$ that keeps tracks on how many times the secret key was changes. That is, the key is obtained now a modified procedure $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D_{\mathtt{ch}}, \mathtt{ch})$, where each of secret key blocks $D'_{\mathtt{ch},i}$ is computed as below:

$$D'_i \leftarrow \mathcal{H}\left(i + \ell \cdot \mathtt{ch}, D_{\mathtt{ch},\sigma(i,1)}, \ldots, D_{\mathtt{ch},\sigma(i,d)}\right) \ ,$$

for $D_{\mathtt{ch},i}$ being potentially changed disk data and $i = 1, \ldots, \ell$. We start with $\mathtt{ch}$ set to 0. In such case we have

$$\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D_0, 0) = \mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D) \ .$$

According to the properties of random oracle, for $j, j' \in \mathbb{N}$ and $j \neq j'$:

$$\mathcal{H}\left(i + j\ell, D_{j,\sigma(i,1)}, \ldots, D_{j,\sigma(i,d)}\right)_{i=1}^{\ell} \text{ and } \mathcal{H}\left(i + j' \cdot \ell, D_{j',\sigma(i,1)}, \ldots, D_{j',\sigma(i,d)}\right)_{i=1}^{\ell}$$

Key update procedure

Input: Two sets of data $D_{\mathtt{ch}-1,0}, \ldots, D_{\mathtt{ch}-1,\ell-1}$ and $D_{\mathtt{ch},0}, \ldots, D_{\mathtt{ch},\ell-1}$. We interpret the former as old data and the later as new data. The prover keeps enumerator $\mathtt{ch} \in \{0,1\}^{\log \ell}$ that shows how many times the public key has been changed. The enumerator is incremented every time data change. The prover is also equipped with a secret key $D'_{\mathtt{ch}-1}$ obtained from Disperse function ran on $D_{\mathtt{ch}-1,0}, \ldots, D_{\mathtt{ch}-1,\ell-1}$ and the corresponding public key $\mathtt{pk}_{\mathtt{ch}-1}$.
Output: A new public key $\mathtt{pk}_{\mathtt{ch}} \in \{0,1\}^n$.

Obtaining a new key

Run $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D_{\mathtt{ch}}, \mathtt{ch})$ as in Figure 4.11, assign the output to $D'_{\mathtt{ch}} = (D'_{\mathtt{ch},0}, \ldots, D'_{\mathtt{ch},\ell-1}) \in \{0,1\}^{n\ell}$.

Legalizing a new public key

Create a new public key Compute a new public key $\mathtt{pk}_{\mathtt{ch}}$ by running $\mathsf{makeTree}(D'_{\mathtt{ch}})$ algorithm (see Figure 4.2). Public key $\mathtt{pk}_{\mathtt{ch}}$ gets the value of the root of computed tree.

Confirm a public key Run non-interactive signature scheme on a Merkle tree defined in Figure 4.9 on message $\mathtt{pk}_{\mathtt{ch}}$ and secret key $D'_{\mathtt{ch}-1}$.

Figure 4.10.: Key update procedure

are independent random variables for probability taken over the choice of D.

Denote by $D_{\mathtt{ch}-1}$ disk data that are subject to change and by $D'_{\mathtt{ch}-1}, \mathtt{pk}_{\mathtt{ch}-1}$ the corresponding secret and public keys. Let $D_{\mathtt{ch}}$ denote the new data and $D'_{\mathtt{ch}}$ be the key derived from it. The new public key is obtained by computing the root c of a Merkle tree with $D'_{\mathtt{ch}}$ as values at leaves. The public key is set to $\mathtt{pk}_{\mathtt{ch}} = c$.

Legalizing a new public key Legalizing a new key is done by broadcasting it along with a signature that confirms its authenticity. That is, we use the non-interactive signature scheme presented in Section 4.7. In this scheme, instead of triple

$$\left((m,r), \mathcal{H}^{[k(n)]}(m,r), (\mathsf{p}_i)_{i\in\mathbf{K}}\right),$$

prover $\mathcal{P}$ sends

$$\sigma = \left((\mathtt{pk}_{\mathtt{ch}},r), \mathcal{H}^{[k(n)]}(\mathtt{pk}_{\mathtt{ch}},r), (\mathsf{p}_i)_{i\in\mathbf{K}}\right),$$

along with $\mathtt{pk}_{\mathtt{ch}}$, where $\mathtt{pk}_{\mathtt{ch}}$ is the new public key. What is worth emphasizing is fact that the signature for $\mathtt{pk}_{\mathtt{ch}}$ is derived from the old key $D'_{\mathtt{ch}-1}$. Analogously, σ is verified by using $\mathtt{pk}_{\mathtt{ch}-1}$. Security of the new public key follows directly from the security of the signature scheme. Obtaining and legalisation of the new public key is described in Figure 4.10.

Implementation of $\mathsf{Disperse}_{\mathcal{G}_\sigma,\mathcal{H}}(D^{\mathtt{ch}}, \mathtt{ch})$**.**

Parameters:

1. d-regular bipartite (ℓ^e, η) disperser $\mathcal{G}_\sigma = (\mathbf{V}^0 \sqcup \mathbf{V}^1, \mathbf{E})$ defined by function $\sigma : [|\mathbf{V}^1|] \times [|\mathbf{E}|] \to [|\mathbf{V}^0|]$, such that $|\mathbf{V}^0| = |\mathbf{V}^1| = \ell$ and $|\mathbf{E}| = d\ell$.
2. random oracle $\mathcal{H} : \{0,1\}^{dn+\log \ell} \to \{0,1\}^n$.

Input: Bitstring $D_{\mathtt{ch}}$ of length $n\ell$, enumerator $\mathtt{ch}$.
Output: Key $D'_{\mathtt{ch}} = D'_{\mathtt{ch},1} \dots D'_{\mathtt{ch},\ell} \in \{0,1\}^{n\ell}$.

Execution:

1. Input $D_{\mathtt{ch}} \in \{0,1\}^{n\ell}$ is divided into ℓ, n-bit long strings: $D_{\mathtt{ch},1}, \dots, D_{\mathtt{ch},\ell}$.
2. Output of $\mathsf{Disperse}$ is divided into blocks as well. We denote these blocks as $D'_{\mathtt{ch},1}, \dots, D'_{\mathtt{ch},\ell}$.
3. For $i = 1, \dots, \ell$: output block $D'_{\mathtt{ch},i}$, gets value $\mathcal{H}\left(i + \ell \cdot \mathtt{ch}, D_{\mathtt{ch},\sigma(i,1)}, \dots, D_{\mathtt{ch},\sigma(i,d)}\right)$,
4. $\mathsf{Disperse}$ returns $D'_{\mathtt{ch}} = D'_{\mathtt{ch},1} \dots D'_{\mathtt{ch},\ell}$.

Figure 4.11.: Implementation of the modified $\mathsf{Disperse}$ function.

Storing old data Another issue which is worth to point out is additional memory demand during establishing a new public key after modification of data $D_{\mathtt{ch}-1}$ (denote the modified data by $D_{\mathtt{ch}}$). This follows from inevitable use of $D_{\mathtt{ch}-1}$ for committing public key for $D_{\mathtt{ch}}$. A simple, yet not very efficient, solution would be to store $D_{\mathtt{ch}-1}$ along with $D_{\mathtt{ch}}$ unless the new key is legalised.

On the other hand, we can assume that $D_{\mathtt{ch}}$ and $D_{\mathtt{ch}-1}$ shares a lot of information, i.e. we suppose that modification of the data comes in parts and it is being changed definitely rarely. To illustrate this assumption by a real-world example we can say that it is much more probable that a single file is deleted than a whole disk is erased. That is, for $D_{\mathtt{ch}-1} = (D_{\mathtt{ch}-1,i})_{i=1}^{\ell}$ and $D_{\mathtt{ch}} = (D_{\mathtt{ch},i})_{i=1}^{\ell}$, $D_{\mathtt{ch}-1}$ and $D_{\mathtt{ch}}$ differ only on small number of indices i. Suppose that $\mathbf{I}$ is a set that contains all such indices. The user who modified data does not need to store the whole $D_{\mathtt{ch}-1}$ along with $D_{\mathtt{ch}}$, but can simply store $D_{\mathtt{ch}}$ along with $(D_{\mathtt{ch}-1,i})_{i\in\mathbf{I}}$. Note, that $(D_{\mathtt{ch}-1,i})_{i\in\mathbf{I}}$ can be removed after the new public key is broadcast.

CHAPTER 5

OPEN PROBLEMS

Weaker adversary, better parameters Security proofs in this thesis highly relied on the Guessing game described in Figure 3.4. Recall, in that game we allow an adversary to leak from both random variables X and $\mathcal{H}$. The first one could be interpreted as a secret key, while the other corresponds to a random oracle.

An interesting research question would be to provide better security parameters for a setting that does not give the adversary so much power. More precisely, the adversary should be able to leak from the secret key, however it is quite unusual that the adversary is allowed to leak from the random oracle also. In the proposed setting, the adversary could, for example, ask the following question: "reveal x, such that $\mathcal{H}(x) = y$". In the real world, getting answer on that would be highly improbable. On the other hand, our model allows the adversary to learn such x. The change of model would require a change in the Guessing game itself what seems like a non-trivial task, taking into account how important the game is for security proofs in this work.

Random Oracle Although it is possible to replace a random oracle by a collision-resistant hash function in any implementation of the proposed protocols, such a replacement does not provide provable security of derived scheme.

In the recent paper, Lindell [Lin15] used a model where all parties are provided with access to a random oracle, yet the oracle is not programmable. The model, originally proposed by Nielsen in [Nie02] is stronger than the standard model, but weaker than ROM. It assumes that no simulator can arbitrarily change an output for an oracle query, the oracle is publicly available. Recall, in this thesis we use programmability of the random oracle to show the security of Disperse function. Along with a non-programmable random oracle, Lindell uses programmable Common Reference String, a standard model tool of achieving common randomness.

The first—and the most important objective from a theoretical point of view—is to provide a BRM-secure key derivation function kdf working in non-programmable ROM without major losses in efficiency and security.

Alternate traversal of a Merkle tree In this work we used the original trespassing algorithm for a Merkle tree. However, there are known other ways to move from a leave to the root, which are more space-efficient and faster, e.g. [JLMS03, Szy04]. Since computational complexity is of the great interest in this work and these traversal methods are not proven to be secure in any leakage-resilient model of operation, future development should consider alternative trespassing of a tree and their security in the BRM.

Secrecy refreshing Merkle tree identification and signature schemes come with inevitable limited number of possible executions. However this obstacle is not a problem since any refreshing needs a single message only. On the other hand, solution deriving a BRM key independent from disk modifications and which does not need public keys updates should be delivered to meet more sophisticated expectations.

Prover-verifier effort balance Proposed solution put almost all effort on the side of the prover: private key is huge compared to very short public key, she is obliged to conduct much more computations than the verifier. We believe that this model suits the real world well—server stores myriads public keys for myriads users each of which stores huge but only one secret key. However, we can picture a situation where more balanced distribution is needed. A scheme allowing arbitrary balancing between length of public and secret key seems an interesting question for further work.

BIBLIOGRAPHY

[AB09] Sanjeev Arora and Boaz Barak. *Computational Complexity: A Modern Approach*. Cambridge University Press, New York, NY, USA, 1st edition, 2009.

[ADN+10] Joël Alwen, Yevgeniy Dodis, Moni Naor, Gil Segev, Shabsi Walfish, and Daniel Wichs. Public-key encryption in the bounded-retrieval model. In Henri Gilbert, editor, *EUROCRYPT*, volume 6110 of *Lecture Notes in Computer Science*, pages 113–134. Springer, 2010.

[ADW09] Joël Alwen, Yevgeniy Dodis, and Daniel Wichs. Leakage-resilient public-key cryptography in the bounded-retrieval model. In Shai Halevi, editor, *Advances in Cryptology - CRYPTO 2009, 29th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2009. Proceedings*, volume 5677 of *Lecture Notes in Computer Science*, pages 36–54. Springer, 2009.

[AM16] Divesh Aggarwal and Ueli Maurer. Breaking RSA generically is equivalent to factoring. *IEEE Trans. Information Theory*, 62(11):6251–6259, 2016.

[BCN89] A.E. Brouwer, A.M. Cohen, and A. Neumaier. *Distance-regular graphs*. Ergebnisse der Mathematik und ihrer Grenzgebiete. Springer, 1989.

[BDK+05] Xavier Boyen, Yevgeniy Dodis, Jonathan Katz, Rafail Ostrovsky, and Adam Smith. Secure remote authentication using biometric data. In Ronald Cramer, editor, *Advances in Cryptology - EUROCRYPT 2005, 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Aarhus, Denmark, May 22-26, 2005, Proceedings*, volume 3494 of *Lecture Notes in Computer Science*, pages 147–163. Springer, 2005.

[BDK+11] Boaz Barak, Yevgeniy Dodis, Hugo Krawczyk, Olivier Pereira, Krzysztof Pietrzak François-Xavier Standaert, and Yu Yu. Leftover hash lemma, revisited. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 2011.

[Bel81] Alexander Graham Bell. US244426 (a) - telephone circuit, 1881.

[Ber12] Daniel J. Bernstein. Optimization failures in SHA-3 software. `http://cr.yp.to/hash/sha3opt-20120104.pdf`, 2012.

[BH05] Boaz Barak and Shai Halevi. A model and architecture for pseudo-random generation with applications to /dev/random. In Vijay Atluri, Catherine Meadows, and Ari Juels, editors, *Proceedings of the 12th ACM Conference on Computer and Communications Security, CCS 2005, Alexandria, VA, USA, November 7-11, 2005*, pages 203–212. ACM, 2005.

[BKKV10] Zvika Brakerski, Yael Tauman Kalai, Jonathan Katz, and Vinod Vaikuntanathan. Overcoming the hole in the bucket: Public-key cryptography resilient to continual memory leakage. In *51th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA*, pages 501–510. IEEE Computer Society, 2010.

[BKR16] Mihir Bellare, Daniel Kane, and Phillip Rogaway. Big-key symmetric encryption: Resisting key exfiltration. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part I*, volume 9814 of *Lecture Notes in Computer Science*, pages 373–402. Springer, 2016.

[BMV05] Lejla Batina, Nele Mentens, and Ingrid Verbauwhede. Side-channel issues for designing secure hardware implementations. In *11th IEEE International On-Line Testing Symposium (IOLTS 2005), 6-8 July 2005, Saint Raphael, France*, pages 118–121. IEEE Computer Society, 2005.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*, CCS '93, pages 62–73, New York, NY, USA, 1993. ACM.

[Bra16] Statistic Brain. Youtube company statistics. `http://www.statisticbrain.com/youtube-statistics/`, 2016.

[BST03] Boaz Barak, Ronen Shaltiel, and Eran Tromer. True random number generators secure in a changing environment. In Colin D. Walter, Çetin Kaya Koç, and Christof Paar, editors, *Cryptographic Hardware and Embedded Systems - CHES 2003, 5th International Workshop, Cologne, Germany, September 8-10, 2003, Proceedings*, volume 2779 of *Lecture Notes in Computer Science*, pages 166–180. Springer, 2003.

[CFPZ09] Céline Chevalier, Pierre-Alain Fouque, David Pointcheval, and Sébastien Zimmer. Optimal randomness extraction from a Diffie-Hellman element. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 572–589. Springer, 2009.

[CGH04] Ran Canetti, Oded Goldreich, and Shai Halevi. The random oracle methodology, revisited. *J. ACM*, 51(4):557–594, 2004.

[Cod14] Codenomicon. The heartbleed bug. `http://heartbleed.com`, 2014.

[CT91] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory*. Wiley-Interscience, New York, NY, USA, 1991.

[CW07] Brian Chess and Jacob West. *Secure Programming with Static Analysis*. Addison-Wesley Professional, first edition, 2007.

[DCLW06] Giovanni Di Crescenzo, Richard Lipton, and Shabsi Walfish. Perfectly secure password protocols in the bounded retrieval model. In *Proceedings of the Third Conference on Theory of Cryptography*, TCC'06, pages 225–244, Berlin, Heidelberg, 2006. Springer-Verlag.

[DDK+16] Konrad Durnoga, Stefan Dziembowski, Tomasz Kazana, Michal Zajac, and Maciej Zdanowicz. Bounded-retrieval model with keys derived from private data. In Kefei Chen, Dongdai Lin, and Moti Yung, editors, *Information Security and Cryptology - 12th International Conference, Inscrypt 2016, Beijing, China, November 4-6, 2016, Revised Selected Papers*, volume 10143 of *Lecture Notes in Computer Science*, pages 273–290. Springer, 2016.

[DDN15] Ivan Damgård, Frédéric Dupuis, and Jesper Buus Nielsen. On the orthogonal vector problem and the feasibility of unconditionally secure leakage-resilient computation. In Anja Lehmann and Stefan Wolf, editors, *Information Theoretic Security - 8th International Conference, ICITS 2015, Lugano, Switzerland, May 2-5, 2015. Proceedings*, volume 9063 of *Lecture Notes in Computer Science*, pages 87–104. Springer, 2015.

[DGH+04] Yevgeniy Dodis, Rosario Gennaro, Johan Håstad, Hugo Krawczyk, and Tal Rabin. Randomness extraction and key derivation using the cbc, cascade and HMAC modes. In Matthew K. Franklin, editor, *Advances in Cryptology - CRYPTO 2004, 24th Annual International CryptologyConference, Santa Barbara, California, USA, August 15-19, 2004, Proceedings*, volume 3152 of *Lecture Notes in Computer Science*, pages 494–510. Springer, 2004.

[DHLW10] Yevgeniy Dodis, Kristiyan Haralambiev, Adriana López-Alt, and Daniel Wichs. Cryptography against continuous memory attacks. In *51th Annual*

IEEE Symposium on Foundations of Computer Science, FOCS 2010, October 23-26, 2010, Las Vegas, Nevada, USA, pages 511–520. IEEE Computer Society, 2010.

[Die12] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2012.

[DKK+12] Yevgeniy Dodis, Bhavana Kanukurthi, Jonathan Katz, Leonid Reyzin, and Adam Smith. Robust fuzzy extractors and authenticated key agreement from close secrets. *IEEE Transactions on Information Theory*, 58(9):6207–6222, 2012.

[DKW11] Stefan Dziembowski, Tomasz Kazana, and Daniel Wichs. Key-evolution schemes resilient to space-bounded leakage. In Phillip Rogaway, editor, *CRYPTO*, volume 6841 of *Lecture Notes in Computer Science*, pages 335–353. Springer, 2011.

[DLWW11] Yevgeniy Dodis, Allison B. Lewko, Brent Waters, and Daniel Wichs. Storing secrets on continually leaky devices. In Rafail Ostrovsky, editor, *IEEE 52nd Annual Symposium on Foundations of Computer Science, FOCS 2011, Palm Springs, CA, USA, October 22-25, 2011*, pages 688–697. IEEE Computer Society, 2011.

[DORS08] Yevgeniy Dodis, Rafail Ostrovsky, Leonid Reyzin, and Adam Smith. Fuzzy extractors: How to generate strong keys from biometrics and other noisy data. *SIAM J. Comput.*, 38(1):97–139, 2008.

[DP07] Stefan Dziembowski and Krzysztof Pietrzak. Intrusion-resilient secret sharing. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*, pages 227–237. IEEE Computer Society, 2007.

[DP08] Stefan Dziembowski and Krzysztof Pietrzak. Leakage-resilient cryptography. In *FOCS*, pages 293–302. IEEE Computer Society, 2008.

[DS05] Yevgeniy Dodis and Adam Smith. Entropic security and the encryption of high entropy messages. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 556–577. Springer, 2005.

[DY13] Yevgeniy Dodis and Yu Yu. Overcoming weak expectations. In Amit Sahai, editor, *Theory of Cryptography - 10th Theory of Cryptography Conference, TCC 2013, Tokyo, Japan, March 3-6, 2013. Proceedings*, volume 7785 of *Lecture Notes in Computer Science*, pages 1–22. Springer, 2013.

[Dzi06] Stefan Dziembowski. Intrusion-resilience via the bounded-storage model. In Shai Halevi and Tal Rabin, editors, *TCC*, volume 3876 of *Lecture Notes in Computer Science*, pages 207–224. Springer, 2006.

[EG85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[Eve15] Rose Eveleth. How many photographs of you are out there in the world? `https://www.theatlantic.com/technology/archive/2015/11/how-many-photographs-of-you-are-out-there-in-the-world/413389/`, 2015.

[FFS88] Uriel Feige, Amos Fiat, and Adi Shamir. Zero-knowledge proofs of identity. *J. Cryptology*, 1(2):77–94, 1988.

[FKPR10] Sebastian Faust, Eike Kiltz, Krzysztof Pietrzak, and Guy N. Rothblum. Leakage-resilient signatures. In Daniele Micciancio, editor, *Theory of Cryptography, 7th Theory of Cryptography Conference, TCC 2010, Zurich, Switzerland, February 9-11, 2010. Proceedings*, volume 5978 of *Lecture Notes in Computer Science*, pages 343–360. Springer, 2010.

[FPS12] Sebastian Faust, Krzysztof Pietrzak, and Joachim Schipper. Practical leakage-resilient symmetric cryptography. In Emmanuel Prouff and Patrick Schaumont, editors, *Cryptographic Hardware and Embedded Systems - CHES 2012 - 14th International Workshop, Leuven, Belgium, September 9-12, 2012. Proceedings*, volume 7428 of *Lecture Notes in Computer Science*, pages 213–232. Springer, 2012.

[FS87] Amos Fiat and Adi Shamir. How to prove yourself: Practical solutions to identification and signature problems. pages 186–194. Springer-Verlag, 1987.

[GGOQ98] Henri Gilbert, Dipankar Gupta, Andrew M. Odlyzko, and Jean-Jacques Quisquater. Attacks on Shamir's "RSA for paranoids". *Inf. Process. Lett.*, 68(4):197–199, 1998.

[GM84] Shafi Goldwasser and Silvio Micali. Probabilistic encryption. *J. Comput. Syst. Sci.*, 28(2):270–299, 1984.

[Gol04] Oded Goldreich. *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.

[GR10] Shafi Goldwasser and Guy N. Rothblum. Securing computation against continuous leakage. In Tal Rabin, editor, *Advances in Cryptology - CRYPTO 2010, 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings*, volume 6223 of *Lecture Notes in Computer Science*, pages 59–79. Springer, 2010.

[GR15] Shafi Goldwasser and Guy N. Rothblum. How to compute in the presence of leakage. *SIAM J. Comput.*, 44(5):1480–1549, 2015.

[Gro04] Jens Groth. *Honest Verifier Zero-Knowledge Arguments Applied*. PhD thesis, University of Århus, Denmark, October 2004.

[Gro06] Jens Groth. Simulation-Sound NIZK Proofs for a Practical Language and Constant Size Group Signatures. In Xuejia Lai and Kefei Chen, editors, *ASIACRYPT 2006*, volume 4284, pages 444–459, Shanghai, China, December 3–7, 2006. Springer, Heidelberg.

[GUV09] Venkatesan Guruswami, Christopher Umans, and Salil P. Vadhan. Unbalanced expanders and randomness extractors from Parvaresh–Vardy codes. *J. ACM*, 56(4), 2009.

[GW11] Craig Gentry and Daniel Wichs. Separating succinct non-interactive arguments from all falsifiable assumptions. In Lance Fortnow and Salil P. Vadhan, editors, *Proceedings of the 43rd ACM Symposium on Theory of Computing, STOC 2011, San Jose, CA, USA, 6-8 June 2011*, pages 99–108. ACM, 2011.

[HerBC] Herodotus. *Histories*. 450-420BC.

[Hig87] Harold Joseph Highland. Tempest over leaking computers. *Computers & Security*, 6(6):457–458, 1987.

[HJK+16] Dennis Hofheinz, Tibor Jager, Dakshita Khurana, Amit Sahai, Brent Waters, and Mark Zhandry. How to generate and use universal samplers. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 715–744, 2016.

[HL02] Michael Howard and David E. Leblanc. *Writing Secure Code*. Microsoft Press, Redmond, WA, USA, 2nd edition, 2002.

[HLW06] Shlomo Hoory, Nathan Linial, and Avi Wigderson. Expander graphs and their applications. *Bull. Amer. Math. Soc. (N.S)*, 43:439–561, 2006.

[HSH+08] J. Alex Halderman, Seth D. Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A. Calandrino, Ariel J. Feldman, Jacob Appelbaum, and Edward W. Felten. Lest we remember: Cold boot attacks on encryption keys. In Paul C. van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium, July 28-August 1, 2008, San Jose, CA, USA*, pages 45–60. USENIX Association, 2008.

[JLMS03] Markus Jakobsson, Frank Thomson Leighton, Silvio Micali, and Michael Szydlo. Fractal merkle tree representation and traversal. In Marc Joye, editor, *Topics in Cryptology - CT-RSA 2003, The Cryptographers' Track at the RSA Conference 2003, San Francisco, CA, USA, April 13-17, 2003, Proceedings*, volume 2612 of *Lecture Notes in Computer Science*, pages 314–326. Springer, 2003.

[Ker83] Auguste Kerckhoffs. La cryptographie militaire. *Journal des sciences militaires*, IX:5–83, 1883.

[KKS11] Yael Tauman Kalai, Bhavana Kanukurthi, and Amit Sahai. Cryptography with tamperable and leaky memory. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 373–390. Springer, 2011.

[KL07] Jonathan Katz and Yehuda Lindell. *Introduction to Modern Cryptography (Chapman & Hall/Crc Cryptography and Network Security Series)*. Chapman & Hall/CRC, 2007.

[KM15] Neal Koblitz and Alfred J. Menezes. The random oracle model: a twenty-year retrospective. *Des. Codes Cryptography*, 77(2-3):587–610, 2015.

[Koc96] Paul C. Kocher. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Advances in Cryptology - CRYPTO '96, 16th Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 1996, Proceedings*, pages 104–113, 1996.

[Lin15] Yehuda Lindell. An efficient transform from sigma protocols to NIZK with a CRS and non-programmable random oracle. In Yevgeniy Dodis and Jesper Buus Nielsen, editors, *Theory of Cryptography - 12th Theory of Cryptography Conference, TCC 2015, Warsaw, Poland, March 23-25, 2015, Proceedings, Part I*, volume 9014 of *Lecture Notes in Computer Science*, pages 93–109. Springer, 2015.

[Lin17] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. *Electronic Colloquium on Computational Complexity (ECCC)*, 24:112, 2017.

[Mer79] Ralph Charles Merkle. *Secrecy, Authentication, and Public Key Systems.* PhD thesis, Stanford, CA, USA, 1979. AAI8001972.

[Mer88] Ralph Charles Merkle. A digital signature based on a conventional encryption function. In Carl Pomerance, editor, *Advances in Cryptology — CRYPTO '87*, volume 293 of *Lecture Notes in Computer Science*, pages 369–378. Springer Berlin Heidelberg, 1988.

[MM05] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems*. Monographs in Computer Science. Springer Science+Business Media, Inc., 2005.

[Nat01] National Institute of Standards and Technology. *Announcing the Advanced Encryption Standard (AES)*. Defense Technical Information Center, 2001.

[Nie02] Jesper Buus Nielsen. Separating random oracle proofs from complexity theoretic proofs: The non-committing encryption case. In Moti Yung, editor, *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*, volume 2442 of *Lecture Notes in Computer Science*, pages 111–126. Springer, 2002.

[NS12] Moni Naor and Gil Segev. Public-key cryptosystems resilient to key leakage. *SIAM J. Comput.*, 41(4):772–814, 2012.

[NSS+17] Matús Nemec, Marek Sýs, Petr Svenda, Dusan Klinec, and Vashek Matyas. The return of coppersmith's attack: Practical factorization of widely used RSA moduli. In Bhavani M. Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1631–1648. ACM, 2017.

[Pie09] Krzysztof Pietrzak. A leakage-resilient mode of operation. In Antoine Joux, editor, *Advances in Cryptology - EUROCRYPT 2009, 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings*, volume 5479 of *Lecture Notes in Computer Science*, pages 462–482. Springer, 2009.

[Rao09] Anup Rao. Extractors for a constant number of polynomially small min-entropy independent sources. *SIAM J. Comput.*, 39(1):168–194, 2009.

[RSA78] Ron Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[RSW06] Omer Reingold, Ronen Shaltiel, and Avi Wigderson. Extracting randomness via repeated condensing. *SIAM J. Comput.*, 35(5):1185–1209, 2006.

[Sch14] Bruce Schneier. Heartbleed. `https://www.schneier.com/blog/archives/2014/04/heartbleed.html`, 2014.

[Sha48] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal, The*, pages 379–423, 1948.

[SJB+14] David Silver, Suman Jana, Dan Boneh, Eric Yawei Chen, and Collin Jackson. Password managers: Attacks and defenses. In Kevin Fu and Jaeyeon Jung, editors, *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pages 449–464. USENIX Association, 2014.

[Spr16] Tom Spring. Web-based keylogger used to steal credit card data from popular sites. `https://threatpost.com/web-based-keylogger-used-to-steal-credit-card-data-from-popular-sites/121141/`, 2016.

[Sta10] François-Xavier Standaert. Introduction to side-channel attacks. In Ingrid M.R. Verbauwhede, editor, *Secure Integrated Circuits and Systems*, Integrated Circuits and Systems, pages 27–42. Springer US, 2010.

[Szy04] Michael Szydlo. Merkle tree traversal in log space and time. In Christian Cachin and Jan Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, volume 3027 of *Lecture Notes in Computer Science*, pages 541–554. Springer, 2004.

[Ta-02] Amnon Ta-Shma. Almost optimal dispersers. *Combinatorica*, 22(1):123–145, 2002.

[Tre01] Luca Trevisan. Extractors and pseudorandom generators. *J. ACM*, 48(4):860–879, July 2001.

[TV00] Luca Trevisan and Salil Vadhan. Extracting randomness from samplable distributions. In *Proceedings of the 41st Annual Symposium on Foundations of Computer Science*, FOCS '00, pages 32–42, Washington, DC, USA, 2000. IEEE Computer Society.

[Vad12] Salil P. Vadhan. Pseudorandomness. *Foundations and Trends in Theoretical Computer Science*, 7(1-3):1–336, 2012.

[YL13] Yanqing Yao and Zhoujun Li. Overcoming weak expectations via the Rényi entropy and the expanded computational entropy. In Carles Padró, editor, *Information Theoretic Security - 7th International Conference, ICITS 2013, Singapore, November 28-30, 2013, Proceedings*, volume 8317 of *Lecture Notes in Computer Science*, pages 162–178. Springer, 2013.

APPENDIX A

ADDITIONAL PROOFS

A.1. Theorem 5.1 from [ADW09]

Proof of Theorem 3 . The proof goes by contradiction. Given an s-entropic adversary $\mathcal{A} = (\mathcal{A}_1, \mathcal{A}_2)$ which has advantage $\varepsilon(n)$ in $\mathsf{EUG}_{\lambda,n}$ (see: Figure 2.6) we construct a Π scheme attacker $\mathcal{B}$ (see Figure A.1) which has probability polynomial in $\varepsilon(n)$ at breaking the security of the Π scheme (see Figure 2.5). That is, we show that the probability that $\mathcal{B}$ convinces verifier $\mathcal{V}$ during the impersonation stage is polynomially related to the advantage of $\mathcal{A}$.

For an execution of $\mathcal{A}$ let S be the event that $\mathcal{A}$ outputs a signature $(\tilde{a}, \tilde{c}, \tilde{z})$ of a new message m. Then by definition, for a random execution we have that $\Pr(S) = \varepsilon(n)$ is exactly the probability of $\mathcal{A}$ winning in the signature attack game.

Let S_1 (occurring with probability $\varepsilon_1(n)$) be the event that S occurs and, at some point, $\mathcal{A}$ queries the random oracle on $\tilde{x} = (\tilde{a}, m)$. Suppose event S occurs but event S_1 does not, then with overwhelming probability the conversation $(\tilde{a}, \mathcal{H}(\tilde{x}), \tilde{z})$ is not an accepting conversation since otherwise $\mathcal{A}$ could be used to break the soundness of the Π scheme. (In particular $\mathcal{A}$, on input only the public key, could generate a valid third flow message for a first flow of it's choosing without even seeing the challenge from the verifier.) Therefore we have that $\varepsilon_1(n) \leqslant \varepsilon(n) - \eta(n)$ for some negligible function η.

In an execution where S_1 occurs let ρ' be the index of the first random oracle query of the form $(\cdot, m)$. Consider an execution of $\mathcal{A}$ emulated exactly as in the real world except that $\mathcal{B}$ first guesses $\rho \leftarrow [q(n)]$. Let S_2 (occurring with probability $\varepsilon_2(n)$) be the event that S_1 occurs and $\mathcal{B}$ guesses $\rho = \rho'$. Since the view of $\mathcal{A}$ is independent of the value of ρ which is chosen uniformly we have that $\varepsilon_2(n) = \varepsilon_1(n)/q(n) = \frac{\varepsilon(n)-\eta(n)}{q(n)}$.

Now consider an execution of $\mathcal{A}$ emulated by $\mathcal{B}$ as described in Figure A.1. Let S_3 (occurring with probability $\varepsilon_3(n)$) be the event that S_2 occurs and $\mathcal{B}$ does not output

Reduction $\mathcal{B}$

The attacker $\mathcal{B}$ gets input $(\mathtt{params}, \mathtt{pk})$ and oracle access to the leakage oracle $\mathcal{O}$ and to a prover $\mathcal{P}(\mathtt{pk}, \mathtt{sk})$. Her goal is to win the $\mathsf{ID}_{\lambda,n}$ game (see Definition 13).

1. Let $q(n)$ be an upper bound on the number random oracle queries made by $\mathcal{A}$. $\mathcal{B}$ selects a random index $\rho \leftarrow [q(n)]$. $\mathcal{B}$ interacts with $\mathcal{P}(\mathtt{pk}, \mathtt{sk})$ using uniformly random challenges c_i to obtain a larger number of conversations of the form (a_i, c_i, z_i). After this point there is no more interaction between $\mathcal{B}$ and $\mathcal{P}$. $\mathcal{B}$ initializes $\mathcal{A}_1$ with input $(\mathtt{params}, \mathtt{pk})$. Eventually $\mathcal{B}$ also passes the hint `hint` from $\mathcal{A}_1$ to $\mathcal{A}_2$.

2. $\mathcal{B}$ simulates the oracles $\mathcal{S}^{\mathtt{sk}}, \mathcal{O}_{\lambda}^{\mathtt{sk}}$ as well as the random oracle $\mathcal{H}$ for $\mathcal{A}$ as follows:

   Random Oracle Queries (except query ρ) For each random oracle query x if $\mathcal{H}(x)$ has already been defined, output it. Otherwise interpret $x = (m, a)$ and see if list of conversations includes one of the form (a, c_j, z_j). If so then define $\mathcal{H}(x) \stackrel{\text{def}}{=} c_j$, associate the conversation (a, c_j, z_j) with x and remove it from the list. Output c_j. If no conversation matches a then select a random response c, define $\mathcal{H}(x) \stackrel{\text{def}}{=} c$ and output c.

   Signature Queries On input m take the next conversation (a, c, z) from the list and check if $\mathcal{H}(m, a)$ is defined. If so there must be a conversation (a, c', z') associated with (m, a). Respond with signature (a, z'). Otherwise define $H(m, a) \stackrel{\text{def}}{=} c$, associate conversation (a, c, z) with (m, a), remove (a, c, z) from the list and output signature (a, c, z).

   Random Oracle Query ρ When the ρ-th random oracle query $x = (a, m)$ is made, if $\mathcal{H}(x)$ has already been defined then $\mathcal{B}$ outputs `abort1` and terminates. Otherwise $\mathcal{B}$ enters the impersonation stage of Π attack game ID and loses access to the leakage oracle. It starts a fresh interaction with the honest verifier $\mathcal{V}(\mathtt{pk})$. $\mathcal{B}$ interprets x as $x = (a, m)$ and sends a to $\mathcal{V}(\mathtt{pk})$ receiving c in response which it outputs as the response to it's ρ-th query.

   Leakage Queries Queries If a leakage query is made after the ρ-th random oracle query then $\mathcal{B}$ outputs `abort1` and terminates. Otherwise it forwards the leakage query to it's leakage oracle and outputs the response.

3. Eventually $\mathcal{A}_2$ outputs a message m and signature (a, c, z). If the signature is valid and (m, a) was the ρ-th random oracle query then $\mathcal{B}$ sends z to $\mathcal{V}(\mathtt{pk})$ and terminates.

Figure A.1.: Reduction from a Σ-adversary to a Π-adversary.

`abort1` and let S_3' be the event that S_2 occurs but $\mathcal{B}$ does output `abort1`. Then $\Pr(S_2) = \Pr(S_2 \wedge S_3) + \Pr(S_2 \wedge S_3') = \Pr(S_3) + \Pr(S_3')$. If event S_3 occurs then the ρ-th query must have been made by $\mathcal{A}_1$ since $\mathcal{A}_2$ can not make leakage queries. Because $\mathcal{A}$ is an s-entropic adversary this can happen with at most probability 2^{-s}. Thus we have that we have $\varepsilon_2(n) \leqslant \varepsilon_3(n) + 2^{-s}$.

Further we argue that in such an emulation, conditioned on event S_3 occurring, the view of an adversary $\mathcal{A}$ is identically distributed to a real world execution. This follows from two observations. First, all random oracle queries in the emulation are answered consistently with uniformly random and independent responses. Further these are also consistent with the responses to the signature oracle. The second observation is that the responses to $\mathcal{A}$'s signature queries have the same distribution as in a real world execution. In particular the values of a and z used in the signatures are selected independently of the message and according to the honest prover's algorithm for a fresh random challenge.

Therefore we can conclude that $\mathcal{A}$ behaves exactly as in real world executions which implies $\varepsilon_3(n) \geqslant \varepsilon_2(n) - 2^{-s} = \frac{\varepsilon_1(n)}{q(n)} - 2^{-s} = (\varepsilon(n) - \eta(n))\, q(n) - 2^{-s}$ which is polynomial related to $\varepsilon(n)$ for $s \geqslant n$. Note that emulated executions where event S_3 occurs correspond exactly to the executions of $\mathcal{B}$ when it convinces $\mathcal{V}$ during the impersonation stage of the Π attack game. Thus the proof for security with pre-impersonation leakage is complete. $\square$