Monero's Building Blocks
Part 1 of 10 – *Prerequisites*

Bassam El Khoury Seguias

BTC: 3FcVvBZwTUkUrcqJd16RcjR42qT2tDWHWn

ETH: 0xb79Fb9194C8Cc6221368bb70976e18609Ab9AcA8

February 28, 2018

1 Introduction

We divide this part into 6 sections. Section 2 is a qualitative description of digital signature schemes. Section 3 motivates the introduction of hash functions along with some of their desired properties. Section 4 describes a hypothetical ideal random function known as a Random Oracle. Section 5 briefly introduces the notion of Probabilistic Turing Machines that will be needed when studying the security of digital signature schemes. Sections 6 and 7 describe 2 pillars introduced by Poitncheval & Stern to prove the resilience of some digital signature schemes against a forgery attack in the Random Oracle model. In particular, Setion 6 describes a reduction model to facilitate the security analysis of signature schemes. Section 7 states and proves an important lemma known as the *splitting lemma*.

There is one caveat: I assume that the reader is familiar with basic probability theory, modulo arithmetic, as well as some group theoretic concepts including the notions of cyclic groups and finite fields. An introduction to group theory is not included in this article, but the reader can refer to the post entitled *Groups and Finite Fields* for a concise overview. For a more detailed treatment, the reader can refer to e.g., [3].

2 Digital Signature Schemes - A qualitative overview

A digital signature can be thought of as a digital alternative to the traditional handwritten signature. The most important property of a handwritten signature is that it is difficult to forge (at least in principle). Similarly, it is crucial that a digital signature scheme be resilient to forgery *(we will formalize the notion of forgery later on)*. In a handwritten signature scheme, the infrastructure consists mainly of 2 elements: 1) the document or the message to be signed, and 2) the signature. In this scheme, the underlying assumption is that even though the signature is made public, it is extremely difficult (in principle) for anyone other than the signer to reproduce it and apply it on another message or document.

The core of a digital signature scheme, is a mathematical construct known as a *secret* or *private key*. This secret key is specific to a signer: 2 different signers will have 2 different secret keys. To ensure this in practice we usually provide a very large pool of allowed private keys. We then assign a unique element from this pool to each potential signer. Once selected, the private key must never be made public.

In the handwritten case, the signature depended solely on the unique ability or talent of the signer to create it. Similarly, in the digital case the signature will depend in part on the private key (which we can simplistically think of as the digital counterpart to the handwritten signer's unique ability). But the digital signature will usually depend on additional information including some random data as well as some available public information. The important design criteria in a digital signature construct are two-fold:

1. The digital signature must conceal the private key of the signer (i.e., no one can realistically recover the private key from the signature), and

2. Anybody can verify the validity of the signature (e.g., that it originated from a particular signer) using only relevant public information.

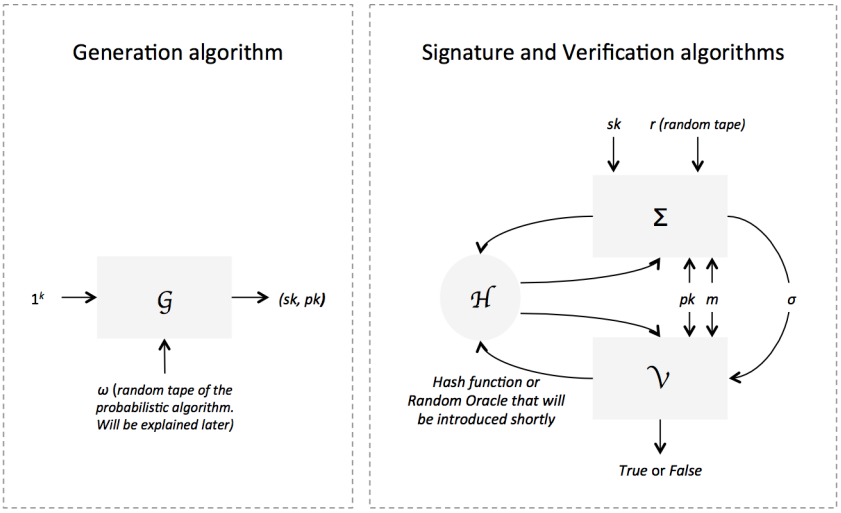
The public information that allows external parties to verify the validity of a digital signature is known as a *public key*. Clearly, the public key must be related to the private key, but it must not divulge any information about it. In general, suppose that private keys are elements of some appropriate finite field, and public keys elements of some appropriate cyclic group G with a generator g, and on which the Discrete Logarithm (DL) problem is thought to be hard. One could define the public key pk associated with the private key sk to be the element of G given by g^{sk}. While it is computationally easy to calculate pk from sk, the reverse is computationally hard. This is because solving for the DL in base g of an element of this cyclic group is thought to be intractable. The hardness of the DL problem provides assurance that sk remains protected.

The above description allows us to define a digital signature scheme the same way as in [4] :*"A user's signature on a message m is a string which depends on m, on public and secret data specific to the user, and -possibly- on randomly chosen data, in such a way that anyone can check the validity of the signature by using public data only. The user's public data are called the public key, whereas his secret data are called the secret key"*. More formaly, a generic digital signature scheme is defined as a set of 3 algorithms:

- **The key generation algorithm** $\mathcal{G}$. It takes as input a security parameter k that ensures cryptographic resilience according to some defined metrics (e.g., k could represent the length in bits of acceptable keys and so the lengthier they are the more resilient the system is). We write 1^k to denote the security parameter input. The algorithm outputs a pair *(pk,sk)* of matching public and secret keys. The key generation algorithm is random as opposed to deterministic.

- **The signing algorithm** Σ. Its input consists of the message m to be signed along with a key pair *(pk,sk)* generated by $\mathcal{G}$. It outputs a digital signature σ on message m signed by the user with private key *sk*. As we will see when we look at specific examples of signing algorithms, the process relies on the generation of random data and is generally treated as a probabilistic algorithm as opposed to a deterministic one.

- **The verification algorithm** $\mathcal{V}$. Its input consists of a signature σ, a message m, and a public key pk. The algorithm verifies if the signature is a valid one (i.e., generated by a user who has knowledge of the private key sk associated with pk). $\mathcal{V}$ is a boolean function that returns *True* if the signature is valid and *False* otherwise. It is a deterministic algorithm as opposed to probabilistic.



An important security concern in any digital signature scheme is how to prevent an adversary with no knowledge of a signer's private key from forging her signature. We define 2 types of attacks that were originally formalized in [1].

1. The **"No-Message" Attack**: The hypothetical attacker or adversary can only have access to the public key of the signer. Moreover, the adversary doesn't have access to any signed message.

2. The **"Known-Message" Attacks**: The hypothetical attacker or adversary has access to the public key of the signer. In addition, the adversary can have access to a subset of message signature pairs $(m, \sigma(m))$. There are 4 scenarios that we consider:

   - **Plain Known-Message Attack**: The hypothetical adversary is granted access to a pre-defined set of message signature pairs $(m, \sigma(m))$ not chosen by him.
   - **Generic Chosen-Message Attack**: The hypothetical adversary can choose the messages that he would like to get signatures on. However, his choice must be done prior to accessing the public key of the signer. The term *generic* underscores the fact that the choice of messages is decoupled from any knowledge about the signer.
   - **Oriented Chosen-Message Attack**: Similar to the Generic Chosen-Message Attack case except that the adversary can choose his messages after he learns the public key of the signer.
   - **Adaptively Chosen-Message Attack**: This is the strongest of all types of attacks. After the adversary learns the signer's public key, he can ask for any message to be signed. Moreover, he can adapt his queries according to previously received $(m, \sigma(m))$ pairs and so is not required to choose messages altogether at once.

In addition to the above modi operandi of a hypothetical adversary, a forgery attack can be of 3 natures:

1. **Total break forgery**: Consists of an adversarial algorithm that learns and discloses the secret key of the signer. Clearly, this is the most dangerous type of forgeries.
2. **Universal forgery**: Consists of an adversarial algorithm that can issue a valid signature on any message.
3. **Existential forgery**: Consists of an adversarial algorithm that can issue a valid signature on *at least* one message.

Whenever we conduct security analysis of digital signature schemes, we endeavour to demonstrate the resilience of the scheme against *existential forgery* under *adaptively chosen-message attack*. By resilience, we mean that the likelihood of finding an adversarial algorithm that can run in polynomial time and succeed in this experiment is negligible. Going forward, we refer to this experiment as **EF-ACM**.

3 Hash functions - Motivation and some properties

Textbook RSA vs. Hashed RSA To motivate hash functions we will rely on [2] and look at a particular signature scheme known as RSA *(the acronym represents the initials of the authors Ron Rivest, Adi Shamir, and Leonard Adleman).* RSA's ability to hide a signer's private key relies on the hardness of the factoring problem. This is different from the hardness of the DL problem introduced earlier. The factoring problem states that it is computationally hard (i.e., no one has yet found an appropriate algorithm that executes in polynomial time) to find the prime factors of a very large integer. We show that the RSA signature scheme in its original form (also known as textbook RSA) satisfies the first desired property of a signature scheme as described earlier, but fails to guarantee the second. This failure will be addressed by introducing the mathematical construct of hash functions.

The textbook RSA signature scheme is a set of 3 algorithms:

- **The key generation algorithm** $\mathcal{G}$ generates the public and private keys of a given user/signer by doing the following:
  - Choose 2 very large prime numbers, say p and q such that $p \neq q$.
  - Let $n = p \times q$ (note that given p and q it is easy to compute n. However, given n, it is very difficult to find p and q. This is the factoring problem).
  - Find n's totient value $\phi(n)$. This value is equal to the number of integers $\leq n$ that are coprime to n. If n is prime, one can easily see that $\phi(n) = n$ - 1. Moreover, when p and q are co-prime, we have $\phi(p \times q) = \phi(p) \times \phi(q)$. Hence we can write $\phi(n) = \phi(p \times q) = \phi(p) \times \phi(q)$ (because p and q are distinct primes and hence are co-prime). This can be further simplified to give $\phi(n)=(p$ - 1) $\times$ (q - 1) (because p and q are prime).
  - Choose an integer $1 < e < \phi(n)$ such that the greatest common divisor $gcd(e, \phi(n)) = 1$. Define (n, e) as the public key of the user.

  - Choose an integer d such that $d \times e \equiv 1 \pmod{\phi(n)}$. In other terms $d \times e = 1 + \alpha \times \phi(n)$ for some integer scalar α. Note that even if e were known, it is computationally hard to find $\phi(n)$ since this requires calculating $(p-1)$ and $(q-1)$ which in turn, requires solving the prime factorization problem. We then define (n, d) as the private key of the user.

- **The signing algorithm** Σ signs a message m with a user's private key (n, d). In this textbook RSA scheme we don't use any additional public information or any randomly generated data to construct the signature. As such, the signing algorithm is deterministic. The message m in this scheme is assumed to be an element of $\mathbb{Z}_n^*$. The algorithm outputs a signature $\sigma \equiv m^d \pmod{n}$. The space of allowed messages is restricted to non-zero integers in this case. What if we want to use this scheme to sign other formats of messages? We will see shortly that this is one of the flexibilities that a hash function introduces.

- **The verfication algorithm** $\mathcal{V}$ verifies if a given signature σ on message m and public key (n, e) is valid or not by checking whether m is equal to $\sigma^e \pmod{n}$. If the equality holds then the algorithm returns $True$. Otherwise, it returns $False$ and the signature is rejected.

One can easily verify that any signature generated by Σ will pass the verification test. Indeed, Σ outputs a signature of the form $\sigma \equiv m^d \pmod{n}$ for a certain private key (n, d) generated by $\mathcal{G}$. This implies that $\sigma^e \equiv (m^d)^e \pmod{n}$. And since $\mathcal{G}$ guarantees that $d \times e \equiv 1 \pmod{\phi(n)}$, one concludes that $\sigma^e \equiv m \pmod{n}$. If all signatures generated by the signing algorithm of a given scheme pass the verification test, we say that the scheme is **correct**. Cleary, textbook RSA is correct.

An important question to ask is whether our signature scheme is resilient against forgery. We will see now that textbook RSA is susceptible to forgery. Later, we remedy the situation by introducing a hash function. Let's look at 2 types of forgeries against textbook RSA:

- *The No-Message Attack*: Recall that in this type of attack, an adversary has access to the signer's public key (n, e) but not to any valid signature on a message. Consider the following: the adversary chooses a random element of $\mathbb{Z}_n^*$ and calls it σ_{forge}. He then forms a message $m = \sigma_{forge}^e \pmod{n}$ and outputs a pair (m, σ_{forge}) as a signature on message m. Note that the verification algorithm $\mathcal{V}$ will compute $\sigma_{forge}^e \pmod{n}$. But this is equal to m as constructed by the adversary. Hence the verifier recognizes (m, σ_{forge}) as a valid signature. The adversary succesfully created a forgery, highlighting the security weakness of this scheme.

- *The Arbitrary Message Attack*: Now suppose that the hypothetical adversary decides to forge a signature of a user with public key (n, e) on an arbitrary message $m \in \mathbb{Z}_n^*$. In the No-Message attack, the forgery was not conducted on an arbitrary message but rather, the message was automatically determined by the procedure. In this case, we allow the adversary to obtain valid signatures from the real signer on 2 messages of his choosing. Consider the following procedure: The adversary first chooses a random message $m_1 \in \mathbb{Z}_n^*$ and computes $m_2 = \frac{m}{m_1}$

$(\text{mod } n)$. The adversary then asks the real signer's signing algorithm to issue a pair of valid signatures σ_1 on mesage m_1 and σ_2 on mesage m_2. He then calculates $\sigma_{forge} = \sigma_1 \times \sigma_2 \ (\text{mod } n)$ and issue the pair (m, σ_{forge}) as a signature on m. The verifier $\mathcal{V}$ computates: $\sigma_{forge}^e \ (\text{mod } n) \equiv (\sigma_1 \times \sigma_2)^e \ (\text{mod } n) \equiv m_1^{ed} \times m_2^{ed} \ (\text{mod } n) \equiv m_1 \times m_2 \ (\text{mod } n) \equiv m \ (\text{mod } n)$ And so the verifier will recognize (m, σ_{forge}) as a valid signature. In this case too, the adversary succesfully created a forgery, highlighting the security weakness of this scheme.

One way of addressing the forgery cases described earlier is by introducing a map $\mathcal{H} : (\textit{message domain}) \rightarrow \mathbb{Z}_n^*$ and apply it to the message to be signed. The signing algorithm Σ would then compute the signature as follows: $\sigma = (\mathcal{H}(m))^d \ (\text{mod } n)$. The verification algorithm would simply check if σ^e is equal to $(\mathcal{H}(m)) \ (\text{mod } n)$. This modified scheme is refered to as the hashed RSA scheme. What are some of the characteristics of $\mathcal{H}$ that will minimize the risk of forgery? We list 2 of them:

1. $\mathcal{H}$ **should exhibit collision resistance**: That means that it should be practically impossible to find 2 distinct messages m_1 and m_2 such that $\mathcal{H}(m_1) = \mathcal{H}(m_2)$. Suppose this were not the case, then if (m_1, σ) is a valid signature, then the hypothetical adversary could find a message $m_2 \neq m_1$ such that $\mathcal{H}(m_1) = \mathcal{H}(m_2)$ and so will successfuly issue a forged signature (m_2, σ).

2. $\mathcal{H}$ **should exhibit pre-image resistance**: That means that it should be practically impossible to invert $\mathcal{H}$ and find x such that $\mathcal{H}(x) = y$ for any given y. We will now see how this property could have prevented the 2 types of forgery attacks in the textbook RSA scheme:

    - *The No Message Attack* case: The forger chooses an arbitrary $\sigma \in \mathbb{Z}_N^*$ as we saw earlier, and calculates $m' = \sigma^e \ (\text{mod } n)$. But now, for (m, σ) to be a succesfull forgery, the adversary still needs to find a message m such that $\mathcal{H}(m) = m'$. And so if $\mathcal{H}$ is difficult to invert, this type of forgery will fail with overwhelming probability.
    - *The Arbitrary Message Attack* case: Following the same logic as in the previous attack, an adversary trying to forge a signature on a message m in the hashed RSA scheme must find $m_1, \ m_2$ s.t. $\mathcal{H}(m) = \mathcal{H}(m_1) \times \mathcal{H}(M_2) \ (\text{mod } n)$ in order to be successful. If $\mathcal{H}$ is difficult to invert, then this forgery will likely fail with overwhelming probability.

To summarize, we motivated the introduction of a map $\mathcal{H}$ in a digital signature scheme such that $\mathcal{H}$ exhibits at least the following 3 characteristics:

1. The domain of $\mathcal{H}$ consists of arbitrary messages of various lengths. We represent the domain by $\{0,1\}^n$, and get $\mathcal{H} : \{0,1\}^n \rightarrow \mathbb{Z}_n^*$. $\{0,1\}^n$ denotes binary strings of length n. The notation $\{0,1\}^*$ denotes binary strings of arbitrary length.

2. $\mathcal{H}$ is collision resistant.

3. $\mathcal{H}$ is pre-image resistant.

Such an $\mathcal{H}$ is refered to as a *hash function* and is likely to limit the cases of successful forgeries. One can also conclude that for all practical purposes, the output of such an $\mathcal{H}$ can be thought of as random-looking.

Hash functions in Non-Interactive Zero Knowledge schemes In 1986, a new paradigm for signature schemes was devised. It consisted in creating an interaction between the signer and the verifier before the verification of the signature took place. The purpose of this interaction was to allow the signer to demonstrate to the verifier that she knows the private key associated with a certain public key without revealing her private key. The idea of demonstrating that you own a piece of information or knowledge without revealing it forms the basis of a set of cryptographic protocols known as *zero-knowledge identification protocols*. As an example we look at the *Schnorr* digital signature scheme:

- **The key generation algorithm** $\mathcal{G}$: Consider a large finite field of prime order p. Let g be a generator of its associated multiplicative subgroup. Choose a random private key $x \in \mathbb{Z}_p^*$ and define its associated public key $y = g^x$. Note that since the Discrete Logarithm problem is intractable on the multiplicative subgroup $\mathbb{Z}_p^*$, it will be hard for any external party to solve for x given y.
- **The signing algorithm** Σ signs a message m as follows:
  - It randomly chooses a value $k \in \mathbb{Z}_p^*$ known as a commitment and computes $r = g^k$.
  - It sends r to the verifier $\mathcal{V}$ who then randomly chooses a value $e \in \mathbb{Z}_p^*$ known as a challenge.
  - The verifier $\mathcal{V}$ then sends e back to Σ who now computes $s = k - e \times x \pmod{p}$.
  - Σ issues a signature on message m in the form of a triplet (r, e, s).
- **The verfication algorithm** $\mathcal{V}$ verifies if r is equal to $g^s \times y^e$. If the equality holds then the algorithm returns $True$. Otherwise, it returns $False$ and the signature is rejected.

One can quickly verify the *correctness* of the *Schnorr* signature scheme. Indeed, let (x, y) be a key pair generated by $\mathcal{G}$ and let (r, e, s) be a signature generated by Σ using x. Then $g^s \times y^e = g^{k-ex} \times (g^x)^e = g^k = r$.

It was shown by Fiat and Shamir that interactive proofs of the kind described in the *Schnorr* scheme above could be replaced by a non-interactive equivalent. The way to do so is to introduce a truly random function that can generate the challeng e originally created by $\mathcal{V}$ in the interactive case. Although hash functions are not truly random, they could play that role in practice. The Fiat-Shamir transformation paved the way to what is know as *Non-Interactive Zero Knowledge Signature Schemes* or *NIZK* for short.

4 The Random Oracle model

It turns out that proving the security (e.g., resilience against forgeability) whenever hash functions are involved is not that straightforward. A setting with a hash function is known as a *standard* model. Rather than opting for no proof at all, a new idealized setting was devised in which cryptographic hash functions are replaced with a utopian

counterpart known as a *random oracle* or *RO* for short. In this environment, it becomes easier to prove the security of various classes of signature schemes. Obviously, a secure scheme in the RO model does not necessarily imply security in the *standard* model where a pre-defined hash function such as SHA-256 is used. Nevertheless, an RO proof allows us to gain a level of confidence higher than if we had no proof at all. There is still an on-going debate regarding the merits of security proofs in the RO model. In what follows, we describe the RO setting and highlight how it differs from the standard model. Our approach follows that of [2].

We can think of the RO as a black box that takes in binary strings of a certain length and outputs binary strings of a possibly different length. No one knows how the box works. Any user can send an input x to RO and receive an output y in return. We say that we query RO with input x. Moreover, RO needs to be consistent. That means that if query x resulted in output y, any subsequent query to RO with the same input x must always result in the same output y. Another way of thinking of the box is as defining a function $\mathcal{H}$ not known to anyone in advance, whose output on a certain query is revealed only when the query is executed. Since $\mathcal{H}$ is not known in advance, it can be considered as a random function. We can interpret the function $\mathcal{H}$ in 2 equivalent ways:

1. Say $\mathcal{H}$ maps n-long bit strings $\{0,1\}^n$ to $l(n)$-long bit strings $\{0,1\}^{l(n)}$ for some appropriate function l. So we have
$$\mathcal{H} : \{0,1\}^n \rightarrow \{0,1\}^{l(n)}$$
$$x \rightarrow \mathcal{H}(x)$$
One way of representing $\mathcal{H}$ is as a very long string where the first $l(n)$ bits represent $\mathcal{H}(00...01)$, the second $l(n)$ bits represent $\mathcal{H}(00...010)$, and so on, where the input is increase by 1 bit every time. Hence, we can think of $\mathcal{H}$ as a $2^n \times l(n)$-bit string. Conversely, any $2^n \times l(n)$-bit string can be thought of as a certain mapping of the form $\mathcal{H} : \{0,1\}^n \rightarrow \{0,1\}^{l(n)}$. We can see that there is a total of $2^{2^n \times l(n)}$ different $\mathcal{H}$ mappings that have the desired input and output lengths. Choosing $\mathcal{H}$ randomly is tantamount to uniformly picking one map among the $2^{2^n \times l(n)}$ different possibilities.

2. Note that randomly choosing $\mathcal{H}$ as per the procedure described above and then storing it somewhere is not realistic. This is due to its sheer size which is exponential in the number of bits. We need to think about what it means for $\mathcal{H}$ to be random using a more pragmatic but equivalent way. We imagine that the black box described earlier generates random outputs for $\mathcal{H}$ on the fly whenever queried. The box would also keep a table of pairs of the form $\{x_i, y_i\}$for all the inputs x_i that have been queried so far, along with their corresponding outputs y_i. If a new query is executed, the box checks if it exists in the table. If not, then it randomly generates an output and adds the pair to the table.

We don't know of the existence of any RO in real life. However the RO model provides a methodology to design and validate cryptographic schemes in the following sense:

- Step 1: Design a scheme and prove its security in the RO model.
- Step 2: Before implementing the scheme in a practical context, instantiate the RO random function $\mathcal{H}$ with a cryptographic hash function $\hat{\mathcal{H}}$ (e.g., SHA-256).

Whenever the scheme queries $\mathcal{H}$ on input x, the practical implementation computes $\hat{\mathcal{H}}(x)$ instead.

Note that the term *instantiate* used above is justified: a practical hash function is an instance in the sense that it is a well defined function and so is not random (as is the case with the RO function). However, one hopes that the hash function behaves well enough so as to maintain the security of the scheme proven under the RO model. But this hope is not scientifically grounded. We mention verbatim the warning of [2]: *"a proof of security for a scheme in the RO model should be viewed as providing evidence that the scheme has no inherent design flaws, but should not be taken as a rigorous proof that any real-world instantiation of the scheme is secure".*

We observe that the RO random function exhibits the desired properties of hash functions we highlighted earlier. In particular *pre-image resistance*, and *collision resistance*

- **Pre-image resistance**: We show that RO behaves in a way similar to functions that are pre-image resistant (also known as *one-way functions*). The subtelty lies in the choice of the verb *behave*, because RO is not a fixed function but rather randomly chosen and not known in advance. So what we will show is that given a polynomial-time probabilistic algorithm $\mathcal{A}$ *(we will discuss this in a bit more details in the Polynomial-Time Turing machine section to follow)* that runs the following experiment:
  - A random $\mathcal{H}$ is chosen as described earlier.
  - A random input $x \in \{0,1\}^n$ is selected.
  - $\mathcal{H}(x)$ is evaluated and assigned to $y \in \{0,1\}^{l(n)}$.
  - $\mathcal{A}$ takes y as an input and outputs $x' \in \{0,1\}^n$ such that $\mathcal{H}(x') = y$.

  then the probability of success of $\mathcal{A}$ is negligible. To see why this is the case, we note that $\mathcal{A}$ succeeds if and only if one of the following 2 situations occur:

  1. $\mathcal{A}$ chooses $x' = x$
  2. $\mathcal{A}$ chooses $x' \neq x$, but RO assigns to $y' = \mathcal{H}(x')$ the same value as $y \equiv \mathcal{H}(x)$

  Suppose that $\mathcal{A}$ can make a total of Q queries to RO, where Q is polynomial in the security parameter k. The security parameter k was briefly introduced earlier in the context of digital signature scheme generation algorithm. It refers to a design parameter such as the length of the output of a hash function, or the length of key pairs in bits. In our case, we restrict k to be $\leq n, l(n)$. Since by construction y is independent of x, then $\mathcal{A}$ knows nothing about x although it knows y. And so the probability that $\mathcal{A}$ chooses a value x' that is equal to x (i.e., situation 1 from above) is equal to:

$$P[\cup_{i=1}^{Q}\{On\ query\ i,\ \mathcal{A}\ chooses\ x'\ s.t.\ x' = x\}]$$

$$\leq \sum_{i=1}^{Q}\ P[\{On\ query\ i,\ \mathcal{A}\ chooses\ x'\ s.t.\ x' = x\}] \leq \frac{Q}{2^n}$$

For situation 2 to materialize, RO must randomly pick a value $y' = \mathcal{H}(x')$ that is equal to $y \equiv \mathcal{H}(x)$ and such that $x \neq x'$. The probability of this happening is equal to:

$$P[\cup_{i=1}^{Q}\{On\ query\ i,\ \mathcal{A}\ selects\ \mathcal{H}(x')\ equal\ to\ \mathcal{H}(x)\ s.t.\ x' = x\}]$$

$$\leq \sum_{i=1}^{Q} P[\{On\ query\ i,\ \mathcal{A}\ selects\ \mathcal{H}(x')\ equal\ to\ \mathcal{H}(x)\ s.t.\ x' = x\}] \leq \frac{Q}{2^{l(n)}}$$

We then conclude that:

$$P[\mathcal{A}\ succeeds] \leq Q \times (\frac{1}{2^n} + \frac{1}{2^{l(n)}}) \leq \frac{2Q}{2^k}, \text{ which is negligible in } k$$

- **Collision resistance**: We show that RO behaves in a way similar to functions that are collision-resistant. By that we mean that for a given polynomial-time probabilistic adversary $\mathcal{A}$ that runs the following experiment:
  - A random $\mathcal{H}$ is chosen as described earlier.
  - $\mathcal{A}$ outputs x and x' such that $x \neq x'$ and such that $\mathcal{H}(x) = \mathcal{H}(x')$.

  the probability of success of $\mathcal{A}$ is negligible. To see why, assume witout loss of generality that $\mathcal{A}$ outputs values x, x' that were queried before the maximum number of Q queries is attained. Moreover, assume that an x is never queried more than once. Since the output of $\mathcal{H}$ is randomly generated for every query (since no query is repeated more than once), we get:

  $$P[\mathcal{A}\ succeeds] = P[\cup_{i,j=1\ i\neq j}^{Q} \mathcal{H}(x_i) = \mathcal{H}(x_j)] = \binom{Q}{2} \times \frac{1}{2^{l(n)}}$$

  Hence we conclude that:

  $$P[\mathcal{A}\ succeeds] = \mathcal{O}(\frac{Q^2}{2^{l(n)}}) \leq \mathcal{O}(\frac{Q^2}{2^k}), \text{ which is negligible in } k$$

Before concluding this section, we highlight the basic idea behind conducting security proofs in the RO model. We contrast it with security proofs in the standard model [2].

Standard real-world model

- $\hat{\mathcal{H}}$ is a pre-defined fixed function.
- We're given a signature scheme $\Pi(r) \equiv (\mathcal{G}(r), \Sigma(r), \mathcal{V})$ that depends on some random parameters r.
- We're also given an adversary $\mathcal{A}(\omega)$, modeled as a polynomial-time probabilistic algorithm *(to be explained shortly)* that attempts to succeed in a certain experiment $\mathcal{E}$, e.g., EF-ACM.
- Given a security parameter k and a negligible function of k (i.e., that decays faster than any polynomial function of k), we define the security with respect to experiment $\mathcal{E}$ as follows:

$$P_{\omega,r}[\mathcal{E}_{\mathcal{A}^{\hat{\mathcal{H}}(\omega)},\Pi^{\hat{\mathcal{H}}(r)}}(k)\ succeeds] \leq \epsilon(k)$$

Note that the above probability is taken over the random parameters, i.e., ω and r. Moreover, both $\mathcal{A}(\omega)$ and $\Pi(r)$ can depend on output from $\hat{\mathcal{H}}$.

Random Oracle model

- $\mathcal{H}$ is a randomly chosen function.
- We're given a signature scheme $\Pi(r) \equiv (\mathcal{G}(r), \Sigma(r), \mathcal{V})$that depends on some random parameters r.
- We're also given an adversary $\mathcal{A}(\omega)$, modeled as a polynomial-time probabilistic algorithm *(to be explained shortly)* that attempts to succeed in a certain experiment $\mathcal{E}$, e.g., EF-ACM.
- Given a security parameter k and a negligible function of k (i.e., that decays faster than any polynomial function of k), we define the security with respect to experiment $\mathcal{E}$ as follows:

$$P_{\omega,r,\mathcal{H}}[\mathcal{E}_{\mathcal{A}^{\mathcal{H}(\omega)},\Pi^{\mathcal{H}(r)}}(k)\ succeeds] \leq \epsilon(k)$$

Here, the probability is not only over ω and r, but also over $\mathcal{H}$. This is because in the RO model, $\mathcal{H}$ is random and not pre-fixed.

The important thing to note in the above comparison is that in the standard model, $\hat{\mathcal{H}}$ is fixed and hence is not taken into account when calculating the probability of success of $\mathcal{A}$ with respect to $\mathcal{E}$. On the other hand, in the RO model, $\mathcal{H}$ is random and so is taken into accounting when computing the probability. Since we cannot be certain that the security result would hold for a particular value of $\mathcal{H}$, we cannot extend the security implication in the RO model to the standard model.

5 Probabilistic Turing machines

The following is a very brief overview of non-deterministic and probabilistic Turing machines. The objective is not to present a detailed analysis of this topic, but rather to gain enough insight and intuition to understand the nature of probabilistic algorithms in digital signature schemes and malevolent adversaries. Most of the material presented here is based on [5]

A Turing machine is a theoretical computer science construct. It can be thought of as a machine or a computer that follows pre-defined rules to read and write symbols (one at a time) chosen from a pre-defined alphabet set. At each time increment, it looks at its current state and the current symbol it is reading. These form its input to the

pre-defined set of rules in order to figure out what action to take. As an example, suppose the machine is in state *happy* with symbol ↑. The pre-defined rule for this pair of inputs consists in modifying ↑ to →, move one step to the left, and update its state to *neutral*.

A Deterministic Turing Machine $\mathcal{M}$ can be formally defined as a 6-tuple in the following way:

$$\mathcal{M} \equiv (Q, \Sigma, i, \phi, A, \delta), \text{ where}$$

- Q is the universe of allowed states which is a finite set.
- Σ is the universe of allowed symbols or alphabet which is also a finite set.
- i is an element of the state-space Q that refers to the initial state.
- ϕ is an element of the alphabet Σ that denotes the blank symbol.
- A is a subset of the alphabet Σ that contains the allowed final states.
- δ is a function $(Q \setminus A) \times \Sigma \to Q \times \Sigma \times \{L, S, R\}$ which takes as input the current state (which must not be an element of the final state set), and the current symbol. It maps them to a well-defined 3-tuple that includes a new state, a new symbol, and a tape movement. Here the tape can either move to the left, right or stay put.

A Non-Deterministic Turing machine can be defined in exactly the same way as its deterministic counterpart with one exception. The relation δ is no longer a function but rather a transition relation that allows an input to be mapped to more than just one output. The transition relation δ is defined a subset of the following cross product:

$$\delta \subseteq [(Q \setminus A) \times \Sigma] \times [Q \times \Sigma \times \{L, S, R\}]$$

The question that remains is how to decide which output to choose if there are many allowed per input. One way of resolving it is by choosing an output drawn from some probability distribution over the range of allowed outputs. This is how a **Probabilistic Turing Machine** operates. We associate with it a random tape ω that encapsulates the probability distribution used by the transition relation δ. In subsequent sections, we model hypothetical adversaries as probabilistic polynomial-time Turing machines (PPT for short) with random tape ω. We write $\mathcal{A}(\omega)$.

6 The reduction model

Recall that proving security of a signature scheme requires proving resilience against forgery. By that we mean resilience against *existential forgery with adaptively chosen-message attack* or EF-ACM. In this setting, $\mathcal{A}(\omega)$ can have access to RO as well as to Σ (the signing algorithm of a given user). However, $\mathcal{A}(\omega)$ does not have access to any user's private key. So $\mathcal{A}(\omega)$ can send any message m to Σ and receive a signature on m as if it were generated by the given user. The objective of $\mathcal{A}(\omega)$ is to be able to create its own signature forgery based on all the queries that it sent to RO and to Σ.

Clearly, $\mathcal{A}(\omega)$ cannot just regurgitate a signature that was created by Σ during one of its earlier queries. It has to create its own. We are then faced with the question of how to approach the problem of proving resilience against EF-ACM. The method we follow was introduced by [4]. The idea is to establish a logical connection between a successful EF-ACM attack and breaking a hard computational problem (usually a discrete logarithm over a well-defined cyclic group). As such, it is considered a conditional proof (i.e., conditional on the intractability of another problem).

To better understand the logic supporting such proofs, let's take another look at the structure of a digital signature scheme. The schemes that we consider have a common skeletal structure. The signing algorithm Σ creates 3 types of parameters that go into building its output signature. The 3 types are:

1. One or more randomly chosen parameters, say $\{r_1, ..., r_k\}, k \geq 1$. These are generated in accordance with Σ's random tape r *(because Σ is non-deterministic, we model it as a PPT Turing machine with a random tape).*

2. One or more outputs of RO on queries that are themselves functions of any of the following:

   - A subset of the random parameters, say $R \subseteq \{r_1, ..., r_k\}$.
   - The message m to be signed.
   - Some other public data, say P (e.g., involving the public key(s) of the signer or a ring of signers as we will see when we introduce ring signature schemes).
   - The secret key(s) of the signer (or a ring of signers), say X.

   The output of query i to the RO can then be represented by $\mathcal{H}[f_i(R_i, m, P_i, X_i)]$, where f_i is some pre-defined function on the relevant parameters, and here i indicates a specific instance of an input parameters.

3. One or more elements that are completely determined by: 1) A subset of the secret key(s) of the signer (or a ring of signers), 2) An element of the first type as described above, and 3) An element of the second type as described above. Call them $\{\alpha_1, ..., \alpha_k\}, k \geq 1$.

For example, we can easily identify the non-interactive version of the *Schnorr*'s signature scheme with this structure. Indeed, the signer randomly chooses a value $k \in \mathbb{Z}_p^*$ known as a commitment and computes g^k, where g is a generator of the group. Then the challenge is calculated as $\mathcal{H}(g^k, m)$. Finally s is calculated as $k - \mathcal{H}(g^k, m) \times x \pmod{p}$, where x denotes the signer's private key. So we can see that:

1. Schnorr has a single random parameter: $r_1 \equiv g^k$.

2. It has a single function f_1 such that $\mathcal{H}[f_1(R_1, m, P_1, X_1)] \equiv \mathcal{H}(r_1, m) = \mathcal{H}(g^k, m)$. And so $R_1 \equiv r_1, P_1 \equiv \emptyset$ and $X_1 \equiv \emptyset$.

3. It has a single fully determined parameter given by $\alpha_1 = s = k - \mathcal{H}(g^k, m) \times x \pmod{p}$.

Any valid signature σ must pass the test of the verification algorithm $\mathcal{V}$. In general, the verifier will conduct a number of queries (say a total of β queries) to RO (or to the hash function in the case of the standard model) and use them to check if a certain relationship holds among some of the signature outputs. In the signature schemes that we consider in this series, we can always identify an equation of the form $g^{a+b\times sk} = C$ where

- g is a generator of the underlying multiplicative cyclic group of the signature scheme.
- a is a quantity calculated by $\mathcal{V}$ that depends on a subset of the signature components, a subset of the β queries that $\mathcal{V}$ sends to RO, and a subset of the RO replies to the β queries.
- b is a quantity calculated by $\mathcal{V}$ that depends on a subset of the signature components, a subset of the β queries that $\mathcal{V}$ sends to RO, and a subset of the RO replies to the β queries that **includes the reply $\mathcal{H}(q_\beta)$ to the last query** q_β.
- C is a quantity calculated by $\mathcal{V}$ that depends on a subset of the signature components, a subset of the β queries that $\mathcal{V}$ sends to RO, and a subset of the RO replies to the β queries, that **does not including the reply $\mathcal{H}(q_\beta)$ to the last query** q_β.

The logic that we follow to prove that a scheme is resilient against EFACM in the RO model, consists in being able to solve for sk in case the scheme were not resilient. That would imply the existence of an adversary $\mathcal{A}(\omega)$ that is able to solve the DL problem on some group. Such a result would contradict the presumed intractability of DL, leading us to conclude that the likelihood of a forgery is negligible.
The question becomes one of linking EFACM with extracting sk. Note that if $log_g C$ were known, one could use $g^{a+b\times sk} = C$ to calculate what the secret key sk is. However, since the DL problem is assumed to be intractable, finding $log_g C$ would be hard. One would need another linear relationship in sk to solve for the secret key.
Suppose a given scheme were not resilient against EFACM, and let σ_{forge_1} be a forgery. We would then have an equation of the form $a + b \times sk = C$. Suppose it can be domonstrated that if the scheme is successfull in generating an initial forgery, then we could replay the attack to generate a second forgery σ_{forge_2}. We also require that the replay of the attack satisfies the following:

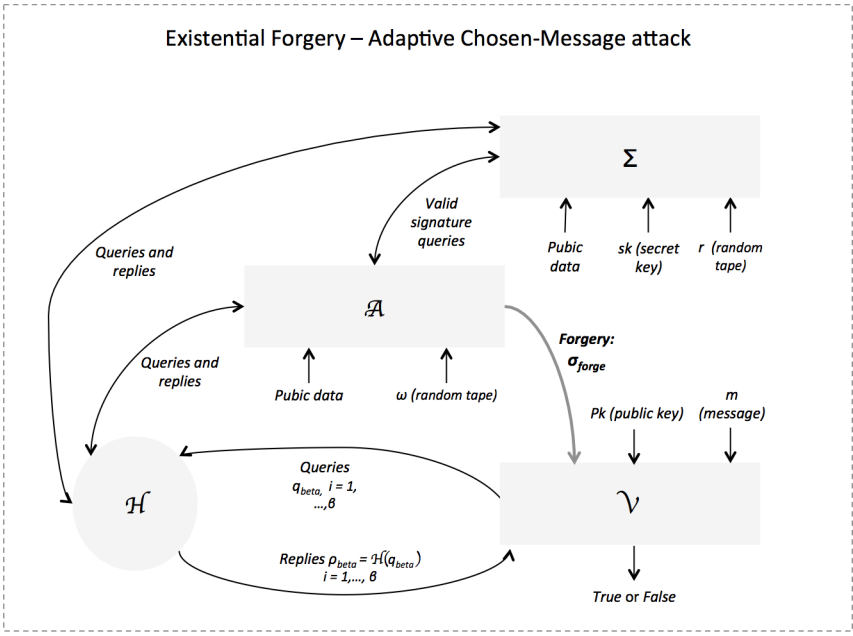
- The second instance of the experiment that generates the second forgery has the same random elements as the first instance of the experiment. Note that there are 2 sources of randomness: one from the random tape ω of the adversary $\mathcal{A}(\omega)$ and another from the random tape r of the signer's signing algorithm $\Sigma(r)$. Here we impose the constraint that the 2 random tapes are maintained between the first and the second instance.
- The queries and the replies that $\mathcal{V}$ sends and receives from RO are the same in the 2 instances, except for the reply on the last query $\mathcal{H}(q_\beta)$.

Under these circumstances, we show when we analyze the different schemes that we can get an equation of the form $a' + b' \times sk = C$ associated with the second forgery. The important thing to note is that since b and b' depend on the reply of RO to the $\beta^t h$

query (which is different for the 2 forgery instances), they will be different from each other. The 2 relationships can then be used to solve for $sk = \frac{a-a'}{b'-b}$ and solve the DL problem.

The reduction model that we are building strives to achieve the above 2 conditions. Let's formalize them. Suppose that an adversary $\mathcal{A}(\omega)$ has a non-negligible probability of success in EF-ACM. As we saw in the RO section previously, this means that:

$$P_{\omega,r,\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\Sigma^{\mathcal{H}}(r)} \; succeeds \; in \; EF-ACM] = \epsilon(k), \text{ for some } \epsilon \text{ non-negligible in k.}$$



A successful forgery corresponds then to a tuple $(\omega^*, r^*, \mathcal{H}^*)$ that allows $\mathcal{A}$ to issue a valid forged signature. It is important to note that $\mathcal{H}^*$ is not a random function anymore but rather a fixed one that took its values after running the first forgery instance. Hence $\mathcal{H}^*$ is a particular instance of $\mathcal{H}$. The 2 constraints regarding the issuance of a second forgery by adversary $\mathcal{A}$ can be summarized in the following equation:

$$P_{\omega,r,\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\Sigma^{\mathcal{H}}(r)} \; succeeds \; in \; EFACM \cap (\mathcal{H}(q_\beta) \neq \mathcal{H}^*(q_\beta)) \mid (\omega^*, r^*, \mathcal{H}^*) \; is \; a \; succesfull \; first \; forgery, \; and \; (\omega = \omega^*), \; (r = r^*), \; (\mathcal{H}(q_i) = \mathcal{H}^*(q_i)) \; for \; i \in \{1, \ldots \beta - 1\}] \equiv \epsilon'(k), \text{ for some } \epsilon' \text{ non-negligible in k.}$$

which we can re-write as

$$P_{\mathcal{H}}[\mathcal{A}(\omega^*)^{\mathcal{H},\Sigma^{\mathcal{H}}(r^*)} \; succeeds \; in \; EFACM \cap (\mathcal{H}(q_\beta) \neq \mathcal{H}^*(q_\beta)) \mid (\omega^*, r^*, \mathcal{H}^*) \; is \; a \; succesfull \; first \; forgery, \; and \; (\mathcal{H}(q_i) = \mathcal{H}^*(q_i)) \; for \; i \in \{1, \ldots \beta - 1\}] \equiv \epsilon'(k)$$

A successful second forgery would then correspond to a tuple $(\omega^*, r^*, \mathcal{H}')$ where here too, $\mathcal{H}'$ is a particular instance of $\mathcal{H}$ and is not a random function anymore.

To make sure that the second time we run the forgery experiment, we use the same random elements that were generated in the first instance, we need to make sure that we use the same tape ω^* for the adversary as well as the same tape r^* for Σ. Making sure that we use the same ω^* is easy since the adversary can record the tape that it used in the first instance and apply it again in the second. The situation is more difficult with replicating the random tape of Σ. This is because, Σ is not controlled by $\mathcal{A}$ and will never collude with it to allow it to forge a signature. Σ will always act honestly and generate new random elements for every instance of an experiment. No constraints can be imposed on its random tape r. This calls for the creation of a new entity $\mathcal{S}(r')$ that we refer to as a simulator with random tape r'. $\mathcal{S}(r')$ would be under the control of $\mathcal{A}$ and hence its random tape r' could be replayed. Clearly, $\mathcal{S}$ will not have access to any secret key, which is the main difference with Σ. Equally important, is that $\mathcal{S}(r')$ must satisfy the following:

$$P_{\omega,r',\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\mathcal{S}^{\mathcal{H}}(r')} \textit{ succeeds in } EF{-}ACM] = P_{\omega,r,\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\Sigma^{\mathcal{H}}(r)} \textit{ succeeds in } EF{-}ACM]$$

If this is satsified, then by showing that $P_{\omega,r',\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\mathcal{S}^{\mathcal{H}}(r')} \textit{ succeeds in } EF{-}ACM]$ is non-negligible we would have showed that $P_{\omega,r,\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\Sigma^{\mathcal{H}}(r)} \textit{ succeeds in } EF{-}ACM]$ is also non-negligible. One way to ensure this equality is by:

1. Making sure that Σ and $\mathcal{S}$ have the same range (i.e., they output signatures taken from the same pool of potential signatures over all possible choices of RO functions and respective random tapes r and r').
2. Σ and $\mathcal{S}$ have indistinguishable probability distribution over this range. *(Refer to [4] for a definition of indistinguishable distributions).* In our case we assume statistical indistinguishability. This means that if we let $(\sigma_1, ..., \sigma_R)$ be an R-tuple of independent signatures, then:

$$P_{r,\mathcal{H}}[\Sigma^{\mathcal{H}}(r) \textit{ returns } (\sigma_1,...,\sigma_R) \textit{ after } R \textit{ queries}] = P_{r',\mathcal{H}}[\mathcal{S}^{\mathcal{H}}(r') \textit{ returns } (\sigma_1, ...,\sigma_R) \textit{ after } R \textit{ queries}]$$

1 and 2 above allow us to write:

$$P_{\omega,r,\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\Sigma^{\mathcal{H}}(r)} \textit{ succeeds in } EF-ACM]$$

$$=$$

$$\sum_{(\sigma_1,...,\sigma_R)}\{P_{\omega,\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H}} \textit{ succeeds in } EF{-}ACM|\ \mathcal{A} \textit{ received } (\sigma_1,...,\sigma_R)]\times P_{r,\mathcal{H}}[\Sigma^{\mathcal{H}}(r) \textit{ returns } (\sigma_1,...,\sigma_R) \textit{ after } R \textit{ queries}]\}$$

$$=$$

$$\sum_{(\sigma_1,...,\sigma_R)}\{P_{\omega,\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H}} \textit{ succeeds in } EF{-}ACM|\ \mathcal{A} \textit{ received } (\sigma_1,...,\sigma_R)]\times P_{r',\mathcal{H}}[\mathcal{S}^{\mathcal{H}}(r') \textit{ returns } (\sigma_1,...,\sigma_R) \textit{ after } R \textit{ queries}]\}$$

$$= P_{\omega,r',\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\mathcal{S}^{\mathcal{H}}(r')} \textit{ succeeds in } EF-ACM]$$

$\mathcal{S}$ also needs to issue valid signatures that pass the verification test. The last step of a verification algorithm is to check whether a certain relationship holds or not. In order to compute the elements for this relationship test, the verifier calculates a number of intermediary values. Intermediary value i is usually a function of the form:

$$f(R_i, P, m, \mathcal{H}(q_i))$$

where $R_i \subset \{r_1, ..., r_n\}$ is a subset of the random parameters generated by the signing algorithm, P is a set of public information including the signer's public key, m is s the message, and $\mathcal{H}(q_i)$ is the i^{th} query sent by the verifier to RO. For the final verification test to be passed, each of these equations must evaluate to a determined value. Any deviation, would result in a failed verification. The signing algorithm Σ uses its knowledge of the signer's secret key to enforce a correct evaluation. However, $\mathcal{S}$ does not have access to the appropriate secret key. And so to make sure that the verification test is satisfied, it will conduct its own random assignment to what otherwise would be calls to RO. We refer to this process as *back-patching*. This methodology guarantees valid signatures, however it requires that $\mathcal{S}$ bypasses RO. From the perspective of $\mathcal{A}$, these assignments are random and it has no way of telling whether they were generated by RO or by another random process. This will not compromise the execution of an experiment as long as the following 2 situations are avoided:

- One of the queries that $\mathcal{S}$ does its own assignment for (i.e., bypassing RO), gets also queried by $\mathcal{A}(\omega)$ directly to RO during execution. Odds are the 2 values assigned by $\mathcal{S}$ and by RO will not match. And so with overwhelming probability, the execution of $\mathcal{A}(\omega)$ will halt. We call this a collision of type 1.
- Suppose $\mathcal{A}(\omega)$ asks $\mathcal{S}$ to sign a certain message m. As part of this process, $\mathcal{S}$ randomly assigns a value to relevant queries $q_i, i \in \{1, ..., n\}$ (for some n). Suppose that at a later time instance, $\mathcal{A}(\omega)$ asks $\mathcal{S}$ to sign some other message m' (it could be equal to m). Here again, $\mathcal{S}$ randomly assigns a value to each of the relevant queries $q'_i, i \in \{1, ..., n\}$. A problem would arise if $q_i = q'_i$ for some i, because the 2 random assignments that $\mathcal{S}$ would issue for these 2 queries will be different with overwhelming probability, and the execution of $\mathcal{A}(\omega)$ will halt. We call this a collision of type 2.

If the probability of occurence of these types of collisions is negligible, the simulator $\mathcal{S}$ can safely do its own random assignment to queries appearing in the verification algorithm without meaningfully affecting the execution. We are then justified in dropping the dependence of $\mathcal{S}^{\mathcal{H}}(r')$ on $\mathcal{H}$ and simply write $\mathcal{S}(r')$.

We close this section with a summary of the outline that we follow in the future to prove resilience against EF-ACM in the RO model:

1. The first step is to assume that there exists a PPT adversary $\mathcal{A}$ successful in EF-ACM in the RO model. In other terms, we assume that:

   $P_{\omega,r,\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\Sigma^{\mathcal{H}}(r)}\ succeeds\ in\ EF-ACM] = \epsilon(k)$, for some ϵ non-negligible in k.

2. The second step is to build a simulator $\mathcal{S}$ such that it:
   - Does not have access to the private key of any signer.
   - Has the same range as Σ (i.e., they output signatures taken from the same pool of potential signatures over all possible choices of RO functions and respective random tapes r and r').
   - Has indistinguishable probability distribution from that of Σ over this range.

Step 1 and the construction in Step 2 imply that:

$P_{\omega,r,'\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\mathcal{S}(r')}\ succeeds\ in\ EF-ACM] = \epsilon(k)$, for some ϵ non-negligible in k.

3. The third step is to show that $P[Col] = \delta(k)$, where δ is negligibe in k. *Col* refers to Collisions of type 1 or 2. This allows us to write:

$$P_{\omega,r,'\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\mathcal{S}(r')} succeeds\ in\ EFACM] =$$

$$P_{\omega,r,'\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\mathcal{S}(r')} succeeds\ in\ EFACM|Col]\times P[Col]+P_{\omega,r,'\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\mathcal{S}(r')} succeeds\ in\ EFACM|\overline{Col}]\times P[\overline{Col}]$$

$$\leq P[Col] + P_{\omega,r,'\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\mathcal{S}(r')} succeeds\ in\ EFACM|\overline{Col}] \times P[\overline{Col}]$$

$$= \delta(k) + P_{\omega,r,'\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\mathcal{S}(r')} succeeds\ in\ EFACM|\overline{Col}] \times (1-\delta(k))$$

And so we can conclude that:

$$P_{\omega,r,'\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\mathcal{S}(r')} succeeds\ in\ EFACM|\overline{Col}] \geq \frac{\epsilon(k)-\delta(k)}{1-\delta(k)},\ \text{(non-negligible in } k)$$

And hence, that:

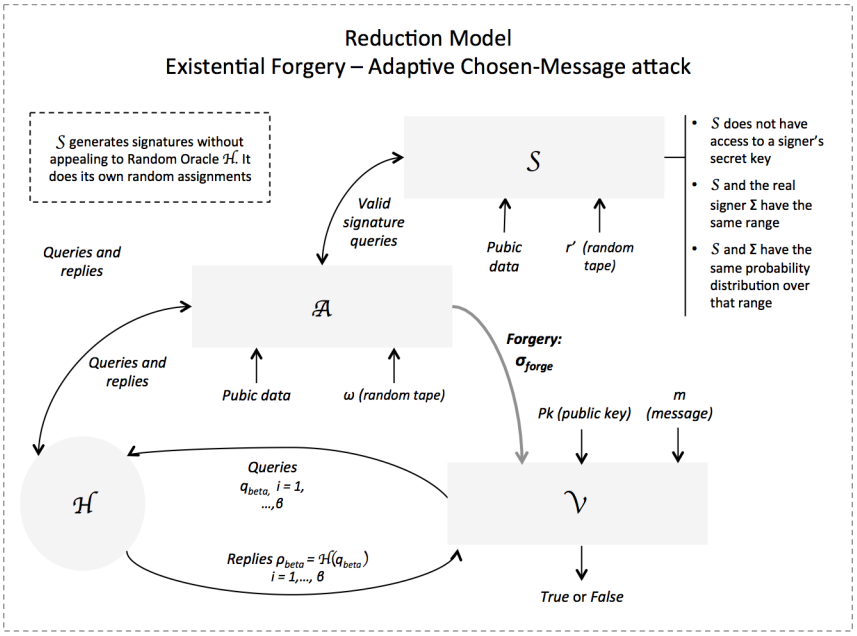
$$P_{\omega,r,'\mathcal{H}}[\mathcal{A}(\omega)^{\mathcal{H},\mathcal{S}(r')} succeeds\ in\ EFACM\ \cap\overline{Col}] \geq \epsilon(k)-\delta(k),\ \text{(non-negligible in } k)$$

4. The fourth step is to prove that if $(\omega^*, r'^*, \mathcal{H}^*)$ is a successful tuple that generated a first EFACM forgery, then the following quantity is non negligible in k:

$$P_{\mathcal{H}}[\mathcal{A}(\omega^*)^{\mathcal{H},\mathcal{S}(r'^*)}\ succeeds\ in\ EFACM\ \cap\ (\mathcal{H}(q_\beta)\neq \mathcal{H}^*(q_\beta))\ |\ (\omega^*,r'^*,\mathcal{H}^*)\ is\ a\ succesfull\ first\ forgery,\ and\ (\mathcal{H}(q_i)=\mathcal{H}^*(q_i))\ for\ i\in\{1,...\beta-1\}] \equiv \epsilon'(k)$$

From this we can obtain a second successful forgery $(\omega^*, r'^*, \mathcal{H}')$ such that $(\mathcal{H}'(q_i) = \mathcal{H}^*(q_i)) for\ i \in \{1, ...\beta - 1\}$ and such that $(\mathcal{H}'(q_\beta) \neq \mathcal{H}^*(q_\beta))$. In order to prove Step 4, we will make use of the *splitting lemma* introduced in [4] that we describe in the following section.

5. The fifth and last step is to use the 2 forgeries obtained earlier to solve an instance of the Discrete Logarithm (DL) problem. This would contradict the intractability of DL on the pre-defined group. This implies that the initial assumption is flawed and that the scheme is resilient against EFACM in the RO model.



7 The Splitting lemma

This lemma, introduced in [4], will play a crucial role in proving Step 4 previously described.

Splitting lemma:

Let A be a subset of a product space $X \times Y$, where each of X and Y are associated with a certain probability distribution.

Let ϵ be such that $P_{(x,y)\sim(X\times Y)}[(x,y) \in A] \geq \epsilon$, where (x,y) is drawn from the joint distribution over X and Y.

For any $0 \leq \alpha < \epsilon$, we define a new subset of $X \times Y$ as follows:
$B = \{(x,y) \in X \times Y s.t. P_{y'\sim Y}[(x,y') \in A] \geq \epsilon - \alpha\}$

Then the following results hold:

i. $P[B] \equiv P_{(x,y)\sim(X\times Y)}[(x,y) \in B] \geq \alpha$

ii. $\forall (x,y) \in B,\ P_{y'\sim Y}[(x,y') \in A] \geq \epsilon - \alpha$

iii. $P[B \mid A] \equiv P_{(x,y)\sim(X\times Y)}[(x,y) \in B \mid (x,y) \in A] \geq \frac{\alpha}{\epsilon}$

Proof:

i. Assume the contrary, i.e., $P_{(x,y)\sim(X\times Y)}[(x,y) \in B] < \alpha$. Then we can write:

$$\epsilon \leq P_{(x,y)\sim(X\times Y)}[(x,y) \in A] =$$

$$P_{(x,y)\sim(X\times Y)}[(x,y)\in A \mid (x,y)\in B]\times P_{(x,y)\sim(X\times Y)}[(x,y)\in B] + P_{(x,y)\sim(X\times Y)}[(x,y)\in A \mid (x,y)\in \overline{B}]\times P_{(x,y)\sim(X\times Y)}[(x,y)\in \overline{B}]\leq$$

$$1\times\alpha + P_{(x,y)\sim(X\times Y)}[(x,y)\in A \mid (x,y)\in \overline{B}]\times 1 < \alpha + (\epsilon-\alpha)\times 1 = \epsilon$$

where the last inequality is derived from the fact that by construction of B, we have $(x,y)\in\overline{B}$ implies that $P_{(x,y)\sim(X\times Y)}[(x,y)\in A] \ < \epsilon-\alpha$

So $\epsilon<\epsilon$, which is a contradiction.

ii. This result is a direct consequence of the definition of B

iii. Baye's rule gives:

$$P_{(x,y)\sim(X\times Y)}[(x,y)\in B \mid (x,y)\in A] = 1 - P_{(x,y)\sim(X\times Y)}[(x,y)\in \overline{B} \mid (x,y)\in A] =$$

$$1 - \frac{P_{(x,y)\sim(X\times Y)}[(x,y)\in A \mid (x,y)\in\overline{B}]\times P_{(x,y)\sim(X\times Y)}[(x,y)\in\overline{B}]}{P_{(x,y)\sim(X\times Y)}[(x,y)\in A]} \geq 1 - \frac{P_{(x,y)\sim(X\times Y)}[(x,y)\in A \mid (x,y)\in\overline{B}]}{P_{(x,y)\sim(X\times Y)}[(x,y)\in A]}$$

$$\geq 1 - \frac{\epsilon-\alpha}{\epsilon} = \frac{\alpha}{\epsilon}$$

As an example, let $X=[1,2,3,4,5,6],\ Y=[1,2,3,4,5,6]$ and let the probability distribution on each of X and Y be the uniform distribution. Moreover, assume that X and Y are independent random variables.

Let $A=\{(1,2),(1,3),(1,5),(1,4),(1,6),(2,4),(2,5),(3,6)\}\subset X\times Y$

Then $P_{(x,y)\sim(X\times Y)}[(x,y)\in A]=\frac{8}{36}=\frac{2}{9}$. So we let $\epsilon=\frac{2}{9}$. We also set $\alpha=\frac{1}{36}<\epsilon$

Lastly, we set $B=\{(x,y)\in X\times Y s.t. P_{y'\sim Y}[(x,y')\in A]\geq\epsilon-\alpha=\frac{7}{36}\}$

Note that if $x\in\{4,5,6\}$, then $P_{(x,y)\sim(X\times Y)}[(x,y)\in A \mid x\in\{4,5,6\}]=0$ by construction of A. So any x that is not part of at least one element of A, can not be part of any element of B. We focus next on the remaining x values:

- If $x=1$:
  $P_{y'\sim Y}[(x,y')\in A \mid x=1] = P_{y'\sim Y}[(x,y')\in\{(1,2),(1,3),(1,4),(1,5),(1,6)\}]=\frac{5}{6}$.
  And since $\frac{5}{6}>\frac{7}{36}=\epsilon-\alpha$, we conclude that the $(x=1)$ section (i.e., all tuples with $x=1$) are members of B.

- If $x=2$: $P_{y'\sim Y}[(x,y')\in A \mid x=2] = P_{y'\sim Y}[(x,y')\in\{(2,4),(2,5)\}]=\frac{2}{6}$. And since $\frac{2}{6}>\frac{7}{36}=\epsilon-\alpha$, we conclude that the $(x=2)$ section (i.e., all tuples with $x=2$) are members of B.

- If $x=3$: $P_{y'\sim Y}[(x,y')\in A \mid x=3] = P_{y'\sim Y}[(x,y')\in\{(3,6)\}]=\frac{1}{6}$. And since

$\frac{1}{6} < \frac{7}{36} = \epsilon - \alpha$, we conclude that the $(x = 3)$ section (i.e., all tuples with $x = 3$) is not a member of B.

In essence, the *splitting lemma* states that if a subset A is *big enough* in a given product space, then it is guaranteed to have many *big enough* sections.

References

[1] S. Goldwasser, S. Micali, and R. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM Journal of Computing*, pages 281–308, 1988.

[2] J. Katz and Y. Lindell. *Introduction to Modern Cryptography.* CRC Press, 2008.

[3] R. Lidl and H. Niederreiter. *Introduction to Finite Fields and their Applications.* Cambridge University Press, 1986.

[4] D. Pointcheval and J. Stern. Security arguments for digital signatures and blind signatures. *Journal of Cryptology*, 2000.

[5] Wikipedia. Non deterministic turing machine.