A basic task in cryptography is ensuring message integrity: When Bob receives a message that is claimed to come from Alice, he wants to be sure that the message originated from Alice, and that it is indeed the message that Alice intended to send. The certificate of integrity usually comes in the form of a tag that is appended to the message that it pertains to. This tag is called a message authentication code and a digital signature in the private and public key settings, respectively.

The honest parties in an authentication mechanism are Alice, who sends out a message together with a tag and Bob, who accepts if the tag is a valid certificate for the message and rejects if not. Alice's distinguishing feature is her secret key. The objective of the adversary Eve is to produce a *forgery*, namely a message-tag pair that Bob accepts as valid.

1 Message authentication codes

A *message authentication code (MAC)* is a pair of circuits (Tag, Ver) describing the tagging and verification procedures: $Tag(K, M)$ produces a tag for a given message M under shared key K, while $Ver(K, M, T)$ checks whether T is a valid tag for M. The functionality requirement is that for every message M, $Ver(K, M, Tag(K, M))$ accepts.

The simplest type of adversary is one that tries to produce a valid message-tag pair without knowing the secret key. A mechanism that protects against it is one that simply outputs the key as a tag:

$$Tag(K, M) = K \qquad Ver(K, M, T) \text{ accepts if } T = K.$$

An adversary cannot produce a forgery without guessing the key.

This MAC doesn't look terribly secure; if Eve manages to intercept the message-tag pair she can change the message part to produce a forgery. In a more realistic attack Eve might observe the tags of one or more messages before she attempts to forge one.

Guided by past experience we can model a MAC attack as a two-phase game. In the learning phase, an adversary interacts with the tagging oracle, obtaining authentication tags $Tag(K, M)$ for messages M of his choice. In the forgery phase he has to produce a message-tag pair (M^*, T^*) that the verifier accepts. The forged message M^* must be different from all the ones that were queried in the learning phase.

A MAC attack is identical to the learning game from lecture 3, so the attacker is defeated by a pseudorandom function:

Theorem 1. *The MAC*

$$Tag(K, M) = F_K(M) \qquad Ver(K, M, T) = \begin{cases} \textit{accept}, & \textit{if } T = F_K(M), \\ \textit{reject}, & \textit{if not}, \end{cases}$$

is $(s, q, 2^{-n} + \varepsilon)$-unforgeable if $F\colon \{0,1\}^m \to \{0,1\}^n$ is (s, q, ε)-hard to learn.

By (s, q, ε)-unforgeable we mean that a size-s, q-query forger succeeds with probability ε.

In many applications of interest the messages are very long. A message may represent a stream of data, in which case it would be desirable to implement the PRF as a streaming algorithm. For

such applications the GGM PRF should be quite suitable in theory, but perhaps not so much in practice as it invokes the underlying pseudorandom generator bit-by-bit. An alternative is to start with a much faster PRF for fixed-length inputs, such as 128-bit AES, which can be viewed as a function $F\colon \{0,1\}^{128} \to \{0,1\}^{128}$, and extend its domain by the CBC (cipher block chaining) transformation:

$$F'(X_1, X_2, \ldots, X_\ell) = F(F(\cdots F(F(X_1) + X_2) \cdots + X_{\ell-1}) + X_\ell).$$

The function F' can be proved to be pseudorandom if F is. Instead of doing this we look into another message length extension method for MACs based on a new cryptographic primitive.

2 Collision-resistant hash functions

Consider a function H that map long inputs into shorter outputs. By the pigeonhole principle, some two inputs must map to the same output, i.e. $H(x) = H(x')$ for some pair of inputs $x \neq x'$. Such a pair is called a *collision*. We say that H is collision-resistant if a collision is computationally hard to find.

There are several extremely efficient proposals of functions that are believed to be collision-resistant for example SHA-256 and SHA-3. No collision has been published for either of them and it is suspected that one will never be found. Nevertheless, it is possible that the designers planted collisions that are known to them in the design, but obfuscated the code so that these collisions are undetectable for the rest of us.

Even if you can live with such suspicions, a modelling issue arises when you try to formalize collision-resistant hashing. Here is a candidate definition: $H\colon \{0,1\}^m \to \{0,1\}^n$ (where $m > n$) is a (s, ε)-collision resistant hash function if for every circuit A of size at most s, the probability that A outputs a collision x, x' for H is at most ε. The trouble with this definition is that such an H cannot exist: The (size zero) circuit that outputs any existing collision breaks H with probability one.

We would like to say that a collision in H is hard to find *given a description of its code*. The designers of H may have made some random choices, which can be summarized in some string K that we can think of as a key. H is then not as a single function but as a collection H_K indexed by the key K. The security requirement is that once K is fixed and made public, collisions in H_K become hard to find.

Definition 2. $H_K\colon \{0,1\}^m \to \{0,1\}^n$ is an (s, ε)-collision-resistant hash function if for every circuit C of size at most s, the probability that $C(K)$ outputs a collision for H_K over the random choice of K is at most ε.

Since the adversary C is bounded in size, it cannot store collision information about H_K for all possible keys K, so it could be plausibly hard for it to locate a collision in a random H_K.

Here is an example based on hardness of discrete logs in base g in the usual setup. The function $H_h\colon \mathbb{Z}_q^2 \to \mathbb{G}$ is given by

$$H_h(x, y) = g^x h^y,$$

where the key is a random element $h \sim \mathbb{G}$. This function shrinks its input length by half, so it must contain (many) collisions. However, if Eve can find a collision $(x, y) \neq (x', y')$ so that $g^x h^y = g^{x'} h^{y'}$ then $h = g^{(y'-y)/(x-x')}$, so she can also find the discrete log of h.[1]

[1] x and x' cannot be equal, because if they are so must be y and y' and the pair is not a collision.

Here is another LWE-based example. The key is an $m \times n$ random matrix A over $\mathbb{Z}_q$. The function $H_A \colon \{0,1\}^m \to \mathbb{Z}_q^n$ is the *modular subset sum* function

$$H_A(x) = xA \bmod q. \tag{1}$$

If $m > n \log q$ collisions in H_A must exist. Now suppose that C finds collisions in H_A. Then C can be turned into this LWE distinguisher: Given (A, y), run $C(A)$ to produce $x \neq x'$ and accept if $(x - x')y$ has magnitude at most bn. If y is of the form $As + e$ then

$$(x - x')y = (xA - x'A)s + (x - x')e = (x - x')e$$

because $xA = x'A$. The entries of $x - x'$ are -1, 0, or 1, so $(x - x')e$ has magnitude at most bn. On the other hand, if y is random and independent of A then $(x - x')y$ is a uniformly random in $\mathbb{Z}_q$ (because $x \neq x'$), so its magnitude will exceed bn except with probability $(2bn + 1)/q$. In summary, the distinguisher tells apart LWE samples from random ones with advantage more than $\varepsilon(1 - (2bn + 1)/q)$.[2]

A collision in any candidate hash function with n bits of output can be found almost certainly in time on the order of $2^{n/2}$ via the so-called *birthday attack*. This works by sampling random and independent inputs $X_1, \ldots, X_\ell$ and looking for a collision among them. The expected number of pairs (X_i, X_j) that are distinct but colliding is the total number of such pairs $\binom{\ell}{2}$ times the probability that a single pair collides, which is at least $2^{-n} - 2^{-m}$. This is on the order of $\ell^2 2^{-n}/2$. When ℓ is on the order of $2^{n/2}$ we expect to see at least one collision, so a hash function can never be more than about $(2^{n/2}, 1/2)$-secure (more generally $(\ell, \ell^2 2^{-n}/2)$-secure assuming $\ell \leq 2^{n/2}$).[3] This is the ideal level of security that designs like SHA-3 aim to achieve.

Just like the stretch of a pseudorandom generator can be increased by composition, so can the *shrinkage* $m - n$ of a hash function.

Lemma 3. *If $H \colon \{0,1\}^m \to \{0,1\}^n$ and $H' \colon \{0,1\}^{m'} \to \{0,1\}^n$ are (s, ε) and $(s+t, \varepsilon')$-secure then*

$$H''_K(z) = H'_K\big(H_K(\textit{some } m \textit{ bits of } z), \textit{remaining } m' - n \textit{ bits of } z\big)$$

is $(s, \varepsilon + \varepsilon')$-secure where t is the size of H.

Proof. Assume we are talking about the first m and last $m' - n$ bits respectively. If (x, y) collide under H'' there are two possibilities. One possibility is that the pair

$$\big(H_K(\text{first } m \text{ bits of } x), \text{last } m' - n \text{ bits of } x\big), \quad \big(H_K(\text{first } m \text{ bits of } y), \text{last } m' - n \text{ bits of } y\big)$$

is a collision under H'_K. If not, then these strings must be identical, so the first m bits of x and the first m bits of y must collide under H_K. So if a circuit C finds a collision for H'', either the first m bits of its output are already a collision under H, or applying H to the first m bits of its output yields a collision for H'. By assumption, the probabilities of these two events are at most ε and ε'. By a union bound, the probability that C finds a collision under H'' is at most $\varepsilon + \varepsilon'$. □

Lemma 3 can be applied iteratively starting with a single hash function H. There are two natural ways to iterate. One is to repeatedly shrink the first m bits, resulting in the Merkle-Damgård construction (Figure 1 (a)). The other one is to start with a length-halving hash function and shrink the input in a tree-like manner. This is called a Merkle tree (Figure 1 (b)). These are mirror images of the pseudorandom generator transformations from lectures 2 and 3. A direct application of the lemma gives the following security guarantees.

[2] It is crucial that the entries of x are *binary* and not from $\mathbb{Z}_q$. In either case, finding a collision amounts to solving $zA = 0$ modulo q with $z \neq 0$. Solving such systems is easy, but finding *short* solutions is hard assuming LWE.

[3] This analysis is incomplete: It lower bounds the expected number of collisions, not the probability of one occurring.

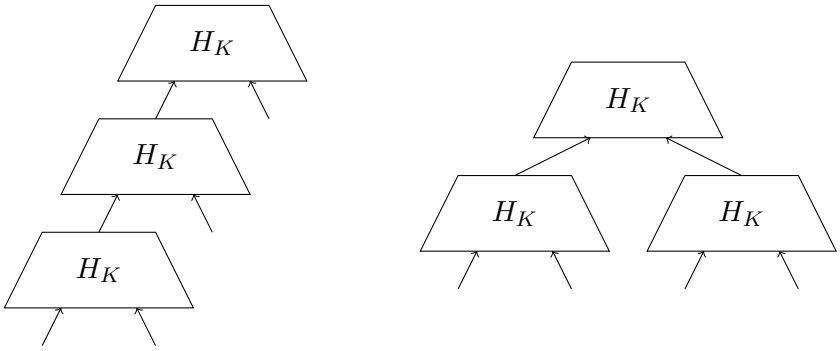

Figure 1: Composition of collision-resistant hash functions. (a) Merkle-Damgård; (b) the Merkle tree.

Theorem 4. *If H is (s, ε)-collision resistant then the Merkle-Damgård construction with ℓ copies of H is $(s - (t-1)\ell, \ell\varepsilon)$-collision resistant. If in addition H is length-halving and $\ell + 1$ is a power of two then the Merkle tree with ℓ copies of H is $(s - t \log \ell, \ell\varepsilon)$-collision resistant.*

A hash of any given string is a short but "computationally unique" identifier of it. This makes hashing particularly suitable for ensuring data integrity. You can take any MAC for short messages, like the one from Theorem 1, and extend it to handle essentially unbounded ones by applying it not to the message itself but to its hash.

Theorem 5. *If (Tag, Ver) is an $(s + t, q, \varepsilon)$-unforgeable for inputs of length n and $H\colon \{0,1\}^m \to \{0,1\}^n$ is an (s, ε')-collision resistant hash of size t, then*

$$Tag'(K, M) = Tag(K, H(M)), \qquad Ver'(K, M, T) = Ver(K, H(M), T)$$

is an $(s, \varepsilon + q\varepsilon')$-unforgeable MAC for message length m.[4]

Proof. Suppose a forger has managed to produce a forgery M for (Tag', Ver'). Then M must be different from all the messages $M_1, \ldots, M_q$ it has queried. One possibility is that $H(M) = H(M_i)$ for some i. then for some i, one of the pairs (M, M_i) is a collision under H. The other possibility is that $H(M)$ is different from all $H(M_i)$, in which case $H(M)$ is a forgery for (Tag, Ver). □

Instantiating H by the Merkle-Damgård construction has implementation advantages for authenticating streaming data, even if the length of the stream is not known in advance. One subtlety that arises when hashing messages (i.e. streams) of varying length is that a collision among two messages of different length can arise. For example, messages MM' and $H(M)M'$ will always collide and an adversary can use this to produce forgeries. An elegant remedy is to append the length of the message as the last block of the message to be tagged. The following claim says that this is secure.

Claim 6. *Let $\{H_i\colon \{0,1\}^i \to \{0,1\}^n\}$ be a collection of (s, ε)-collision resistant hash functions and $H\colon \{0,1\}^{n+\log \ell} \to \{0,1\}^n$ be another $(s + 2t, \varepsilon')$-collision resistant hash for fixed input length. Then the function $H'(x) = H(H_{|x|}(x), |x|)$ is $(s, \varepsilon + \varepsilon')$-collision resistant for messages of any length up to ℓ, where t is the common size of H_i for $i \le \ell$.*

Proof. Suppose x and y collide under H'. If they have the same length i and they do not collide under H_i, then $(H_i(x), i)$ and $(H_i(y), i)$ is a collision for H. Otherwise they have different lengths, so $(H_{|x|}(x), |x|), (H_{|y|}(y), |y|)$ is a collision for H. □

[4]The key of H is public information available to all the parties.

3 Digital signatures

Digital signatures are the public-key variant of message authentication tags. In the public-key setting not only Bob but anyone can verify the authenticity of a signature. Without the knowledge of the secret key Eve should not be able to forge one even after having observed her fair share of messages signed by Alice. This is fairly similar to the guarantees we expect from a physical signature that one puts on a piece of paper.

Definition 7. A *digital signature* consists of three circuits $(Gen, Sign, Ver)$ such that for every message M, $Ver(PK, M, Sign(SK, M))$ accepts with probability one, where (SK, PK) is a key pair sampled by Gen.

The forging game is the same as before, except now the forger also knows the public key PK. Unlike in the case of public-key *encryption*, the forger still needs access to a signing oracle $Sign_{SK}$ as she doesn't know the public key and cannot produce any signatures on her own.

Here is a solution that works against a restricted adversary that makes only one query to the signing oracle. To illustrate it we'll show how it works for a one-bit message $M \in \{0, 1\}$. The building block is a pseudorandom generator G. The secret key consists of two random inputs X_0 and X_1 for G and the public key consists of $Y_0 = G(X_0)$ and $Y_1 = G(X_1)$.

To sign a message M, Alice reveals the value $Sign(SK, M) = X_M$. Bob verifies that G applied to this value yields Y_M; that is $Ver(PK, M, X)$ accepts only if $G(X) = M$. If Eve queries the signing oracle on, say $M = 0$, she does not obtain any information that is relevant for signing $M = 1$; in order to do that she must invert G on Y_1 which is computationally hard.

This scheme extends to longer messages by applying it to each bit of the message separately. To do this, the key generation procedure ought to produce independent keys for every bit of the message to be signed. There are improvements that shorten the keys and provide security against forgers of unbounded query complexity, but signature schemes constructed along these lines are considered quite inefficient and impractical. Nevertheless, one advantage is that their security follows merely from the existence of pseudorandom generators and not from specific assumptions like the hardness of discrete logs or LWE.

Instead we describe a different methodology that exploits the similarity between identification and signatures. It is natural to think of signatures as an extended form of identification: Identification is about Alice proving her identity (i.e., knowledge of her secret key) to Bob, while signatures are about Alice proving to Bob that she wrote some particular message.

In one aspect the security requirement for signatures is weaker than that for identification: Eve must sign a message that is different from all the ones she has seen, so she cannot simply replay an interaction with her signing oracle and submit that as her forgery. The functionality requirement is, however, stronger in that the signature must be produced *non-interactively*. As we discussed last time, non-interactive identification is clearly insecure against eavesdropping.

Schnorr's identification scheme is almost non-interactive: Prover sends the preamble $h = g^R$, verifier challenges with a random C, prover responds with $Y = R + CX$ (where (X, g^X) is the key pair) and verifier accepts if $h \cdot PK^C = g^Y$. All the verifier does in this interaction is sample and send a completely random message. Can't this part be outsourced to the prover? No, because a cheating prover can pretend, say, that C equals zero and pass validation. How about making C part of the public key? The prover can cheat again by picking, say, $h = PK^{-C}$ and $Y = 0$.

4 Random oracles and the Fiat-Shamir heuristic

Fiat and Shamir proposed replacing the verifier's challenge by a *random function* of the preamble h. To make sense of this suggestion we need to talk about the random oracle model.

Let's recall how things worked out for us in the private-key setting. To get CPA-secure encryption, we started with a scheme based on a random function and then replaced it with a pseudorandom one. Secure identification and unforgeable MACs worked out in pretty much the same way. In all three cases, we start with an ideal object—a truly random function—and prove security. We then separately analyze the effect of replacing this ideal random function with a real pseudorandom one.

The random oracle model is a setup in which ideal cryptographic schemes can be studied without thinking about implementation. Each party, including the adversary, is given *oracle access* to the same random function R. Initially, none of the parties have any information about what R is. When Alice makes a query, say $R(\texttt{001})$, she is provided with a random value. When Bob queries say $R(\texttt{101})$, he is given another random value which is independent from the one observed by Alice. But if Alice now queries $R(\texttt{101})$ she observes the same value that Bob just did.

Let's try to define a random oracle variant of the MAC from Theorem 1. A first attempt is to use a truly random function instead of the pseudorandom one:

$$Tag^R(M) = R(M), \quad Ver^R(M,T) = \text{accept if } T = R(M), \qquad R \sim \{0,1\}^m \to \{0,1\}^n.$$

This scheme is perfectly unforgeable assuming that the forger Eve cannot query R. After observing however many MACs she likes, Eve has no information about $R(M)$ for a previously unseen M. The random oracle model requires security even if Eve has access to R. The MAC is then clearly insecure as Alice and Bob share no secret key relative to Eve. Here is a secure variant:

$$Tag^R(K,M) = R(K,M), \quad Ver^R(K,M,T) = \text{accept if } T = R(K,M). \tag{2}$$

Now even if Eve has access to R, the q MACs she observes are all of the form $M_i, R(K,M_i)$ for the same secret key $K \sim \{0,1\}^k$. To produce a forgery on a new message M^* she must guess the value $R(K,M^*)$. Unless she happens to query her oracle on an input that starts with K, her forged tag is independent of $R(K,M^*)$ so it can only be correct with probability 2^{-n}. On the other hand, the MACs she observes are independent of K, so the probability that she makes a query that starts with K is at most $q \cdot 2^{-k}$. We just proved that

Claim 8. *MAC (2) is $(\infty, q, q \cdot 2^{-k} + 2^{-n})$-unforgeable in the random oracle model.*

The security guarantee is statistical: The size of the adversary is only restricted by the number of times it queries the random oracle.

The other private-key constructions that we saw so far are likewise statistically secure in the random oracle model. On the other hand, a famous theorem of Impagliazzo and Rudich says that no statistically secure key exchange protocol exists in the random oracle model. This is why our examples of key exchange and public-key encryption look so different from the protocols in the private-key setting.

The Fiat-Shamir heuristic is a transformation that takes an interactive protocol like Schnorr's and turns it into a non-interactive protocol in the random oracle model. To apply the transformation, the random messages sent by the verifier are replaced by outputs of the random oracle evaluated on the previous messages. Figure 2 (a)-(b) shows the protocol (P^R, V^R) obtained by applying the Fiat-Shamir heuristic to Schnorr's protocol (P,V).

Suppose (P,V) is Schnorr's protocol with long challenges which is secure against eavesdropping. The protocol (P^R, V^R) is no longer secure because it is deterministic. A cheating prover (with oracle

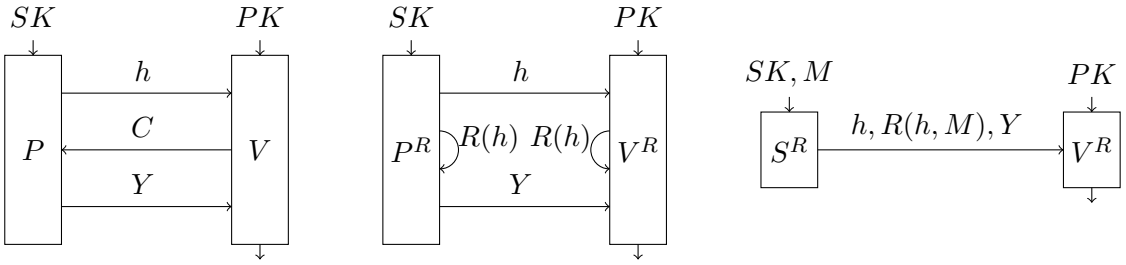

Figure 2: Digital signatures from identification: (a) Schnorr's protocol format; (b) the Fiat-Shamir heuristic; (c) The digital signature scheme.

access to R) can pass validation by replaying an eavesdropped interaction. But this is essentially the only attack that is available to him! If he tries to produce an interaction that starts with a different preamble h^* than the ones he eavesdropped on he is out of luck because the challenge $R(h^*)$ will be independent from all previous messages, looking exactly like the challenge C in Schnorr's protocol. If he can handle the challenge $R(h^*)$ in (P^R, V^R), he could have handled a random challenge C in Schnorr's protocol (P, V) and passed validation there.

This is very close to the security requirement of a signature scheme, which discounts forgeries of messages that were queried by the signing oracle. Indeed, a small upgrade to (P^R, V^R) yields a signature scheme (S^R, V^R) in the random oracle model (see Figure 2 (c)).

Schnorr's signature scheme. The public-private key pair is $SK = x, PK = g^x$ for a random $x \sim \mathbb{Z}_q$. The random oracle R is a function from $\mathbb{G} \times \{0,1\}^m$ to $\mathbb{Z}_q$.

- To sign M, Alice outputs $(h = g^{R'}, C = R(h, M), Y = R' + CX)$ for a random $R' \sim \mathbb{Z}_q$.
- To verify the message-signature pair $M, (h, C, Y)$, Bob checks $C = R(h, M)$ and $g^Y = h \cdot PK^C$.

Theorem 9. *If Schnorr's protocol is (s, q', ε)-secure against eavesdropping then Schnorr's signature scheme is $(s - O(q'^2 \log q), q', q'\varepsilon + (q'^2 + 1)/q)$-unforgeable in the random oracle model.*

To prove this theorem, we need to turn a forger F for Schnorr's signature scheme into a cheating prover P^* for Schnorr's indentification protocol.

In the learning phase, P^* simulates the learning phase of F. The forger F can make two types of queries: Random oracle queries and signing queries. P^* answers random oracle queries by fresh random values and signing queries by eavesdropped identification transcripts.

P^* must be careful to avoid inconsistencies: For example, if F queried his signing oracle at M to obtain (h, C, Y) and then its random oracle at (h, M), it expects to observe the value C. To ensure consistency in such cases, P^* can be implemented like this:

- When F queries R, P^* returns a fresh random value unless F has made this query before.
- When F makes a signing query on M, P^* returns a fresh eavesdropped transcript (h, C, Y) and treats (h, M) as a query that F has made to R with answer C.

Despite P^*'s best efforts two types of inconsistencies can still arise: Some pair of signing queries could collide in M and h (but not C), or F could query R on a pair (M, h) that later arises as part of an eavesdropped transcript. If these inconsistencies are avoided, F's view is identically

distributed to what it would be had it interacted with the real oracles R and S^R. Since the h's in the eavesdropped transcripts are random, the probability of an inconsistency is at most q'^2/q.[5]

After the learning phase, F outputs a forgery $M^*, (h^*, C^*, Y^*)$. P^* would like to turn this forgery into successful validation. There is one problem: In the identification protocol the challenge is not fixed, but it is chosen at random by the verifier V.

The last ingredient is the cut-and-choose trick from Lecture 5: The cheating prover makes an educated guess that at some point the forger should have asked the random oracle about (h^*, M^*) and received C^* as an answer. Instead of furnishing C^* as fresh randomness it obtains it as a challenge from the verifier.

To implement this idea it will be useful to play the eavesdropping game in a slightly more general format. Instead of dividing the game into a learning and validation phase as before, we can allow for validation to start before the learning phase is over. After observing some transcripts, the cheating prover begins his validation attempt with a preamble of his choice, receives a challenge, then does some more learning before coming up with a response. This does not affect the cheating prover's advantage.

Proof of Theorem 9. The cheating prover P^* operates as described with one difference. Initially P^* chooses a random index I between 1 and q'. When F makes its I-th query (h_I, M_I) to R, P^* initiates validation, submitting h_I as preamble and receiving C as a challenge from the verifier V. It replies C to F (and records this query-answer pair). After that, it carries on with learning. When F produces a forgery $M^*, (h^*, C^*, Y^*)$, P^* outputs Y^* as his response if $(h^*, M^*) = (h_I, M_I)$ and fails otherwise.

We consider two cases. If F never queried his random oracle on (h^*, M^*), the chances that his output is a valid forgery is $1/q$ because $R(h^*, M^*)$ is independent of C^*. Otherwise, conditioned on (h^*, M^*) being F's I-th query to R, the transcript between P^* and V in the eavesdropping game is identically distributed[6] to the transcript between F^R and S^R in the forgery game with the forgery in the latter representing the validation attempt in the former. So the two succeed with the same probability. Since the conditioning happens with probability at least $1/q'$, P^* is validated with probability at least $(\varepsilon' - (q'^2 + 1)/q)/q'$ assuming F produces a forgery with probability ε'. □

All that remains to do is to eliminate the random oracle. Your instinct (and mine) should tell you to try and replace it with a pseudorandom function F_K. But pseudorandom functions are not designed to operate in a public-key environment, so what to do with the key? If Eve knows the key K she can easily distinguish F_K from a truly random function and Theorem 9 would have no consequence for the security of the resulting signature scheme.

Constructions in the random oracle model would remain secure in a public-key setting if the oracle was replaced with an obfuscated pseudorandom function. VBB-obfuscation of pseudorandom functions is however impossible. Moreover, there exist protocols that are secure in the random oracle model, but any instantiation of the random oracle by a specific function breaks security.

Nevertheless, Schnorr's signature scheme is believed to be secure in practice when the random oracle is replaced by a sufficiently "random-looking" function like SHA-3.

[5] More precisely, if F makes q_R queries to R and q_S queries to the signing oracle then the probability of a collision is at most $(q_R q_S + \binom{q_S}{2})/q$.

[6] As M^* is part of the forgery, the signing oracle was not queried on it so it couldn't be part of an inconsistency.