Foundations of Cryptography

Notes of lecture No. 10B & 11 (given on June 11 & 18, 1989)

taken by Sergio Rajsbaum

Summary

In this lecture we define unforgeable digital signatures and present such a signature scheme based on the assumption that one-way permutations with trapdoor exist.

1. Introduction.

The idea of a "digital signature" first appeared in Diffie and Hellman's seminal paper, "New Directions in Cryptography" [DH76]. They propose that each user U publishes a "public key", while keeping secret a "secret key". User U's signature for a message m is a value σ depending on m and his public and secret keys such that U can generate σ and anyone can verify the validity of σ using U's public key. However, while knowing U's public key is sufficient to allow one to validate U's signatures, it does not allow one to efficiently forge U's signatures.

The most severe natural attack an enemy can mount on a signature scheme is called *adaptive chosen message attack*, or simply chosen message attack, denoted *CMA*. In this type of attack the enemy is allowed to use U as an "oracle"; he tries to forge a signature after getting from U signatures to messages of his own choice. If there exists no probabilistic polynomial time algorithm that can forge a message in this way, we say that the signature scheme is secure against *CMA*.

There exist several, weaker types of attacks. In *directed chosen message attack*, the enemy is allowed to obtain from U valid signatures for messages $m_1, \ldots, m_t$, chosen based only on U's public key. This attack is nonadaptive: the entire message list is constructed before any signatures are seen. An even weaker type of attack is called *known message attack*. In this type of attack an adversary is only given signatures for messages selected at random.

The reason for requiring security against *CMA*, is that we do not want to ask U to be careful about what he signs, or to sign only meaningful messages of his own choice. We are not to decide what a "meaningful" message is. The user should be able to use the signature scheme freely, and decide which documents he wishes to sign without woring about technical limitations of the scheme itself.

With respect to "breaking" a signature scheme there are also several definitions, we shall consider the strongest: We say that an enemy is able to break the system, or *forge* a signature if he can create in polynomial time and with non-negligible probability, a pair of strings which he has not seen before, consisting of a message and its signature (even if the message has no "meaning" at all).

What we prove to be difficult is *forgery*, and not merely obtaining the secret key s used by the signing algorithm. A signature scheme in which nobody can obtain in reasonable time the signing key, does not necessary prevent an attacker from forging signatures. It could be possible to forge signatures without knowing s, or even to obtain only a crucial part of s, which enables to forge signatures.

The signature scheme presented here is based on the work of Bellare and Micali [BM88]. It has the following important characteristics:

- Forgery is proven to be difficult for a "most general" enemy who can mount a *CMA*.
- The properties we prove about the new signature scheme do not depend in any way on the set of messages which can be signed or on any assumptions about a probability distribution on the message set.
- The scheme is general in the sense that it can be based on generic "hard" problems (other than factoring), yielding a signature scheme that is invulnerable to a *CMA* even if the "hard" problem used *is* vulnerable to a *CMA*.

The way of proving that the signature scheme is secure against *CMA*, will be to show that an enemy cannot break the system unless he can perform some other task, which is assumed to be infeasible. In our case we shall assume the existence of one-way permutations with trapdoor.

In the paper [NY89], Naor and Yung show a stronger result. They prove the existence of signature schemes secure against *CMA*, based on the existence of one-way permutations (without trapdoor).

2. Signature Schemes.

We recall now the basic notions and formal definitions of signature schemes given in the previous lecture.

A *signature scheme* is a triple of algorithms

$$(G,\ S,\ V)$$

with a *security parameter*, n. The *key generating* algorithm G is a probabilistic-polynomial time algorithm that generates ordered pairs $(s,\ v)$ of a *signing key*, s, and a *verification key*, v:

$$G(1^n) = (s,\ v).$$

S is a *signing* probabilistic polynomial time algorithm[1] which produces signatures σ, with respect to a given key s produced by $G(1^n)$, for every message m of length n, taken from some set M:

1 The algorithm S does not have to be probabilistic. Even if it is, it can be transformed into a deterministic one by thinking of s as $s's''$, and including the coin tosses that S needs in s'', while leaving s' as the secret key.

$$S(s,\ m) = \sigma.$$

And *V*, a *verifying* polynomial time algorithm (possibly probabilistic), which serves to check whether σ is a valid signature for *m*, with respect to the key *v*. For every $(s,\ v)$ in the range of $G(1^n)$, and every $m \in M$ of length *n*, if σ is in the range of $S(s,\ m)$, then

$$V(v,\ m,\ \sigma) = 1\ ,$$

otherwise $V(v,\ m,\ \sigma) = 0$.

The last definition for the verifier can be replaced by a probabilistic definition that would allow for a small probability of error: For $(s,\ v)$ in the range of $G(1^n)$, and every $m \in M$ of length *n*, for every $c > 0$, and sufficiently large *n*

$$Pr\left\{V(v,\ m,\ S(s,\ m)) \neq 1\right\} < \frac{1}{n^c},$$

where the probability is taken over the coin tosses of algorithms *G*, *S* and *V* (if any).

To utilize the signature scheme, *U* creates a pair $(s,\ v)$ using the algorithm *G*. He then stores *v* in the public file (*v* appears in the public file under *U*'s name), and keeps *s* secret. When *U* wants to sign a message *m* he executes $S(s,\ m)$ and obtains the string σ which is the signature for *m*. Everybody can verify that σ is a valid signature for *m* by checking that the result of $V(v,\ m,\ \sigma)$ is actually 1.

Security Against *CMA*.

In the scenario of *CMA*, an algorithm *F* that tries to *forge* a signature has access to the signing algorithm *S* with key *s*, denoted S_s (but *F* has no knowledge of the key *s*); it can obtain the signatures with respect to *s*, of any messages it chooses, and use them to try to *forge* a another signature; we denote by F^{S_s} such an execution of algorithm *F*. By forging we mean the ability of *F*, on input *v*, to produce a pair $m,\ \sigma$ (which it has not seen before), such that $V(v,\ m,\ \sigma) = 1$. A signature scheme is *secure against CMA* if for all probabilistic, polynomial-time forgering algorithms *F*, for all $c > 0$, and for all *n* sufficiently large,

$$Pr\left\{F^{S_s}(v)\ \ forges\right\} < \frac{1}{n^c}$$

where the probability is taken over the coin tosses of the algorithms *F*, *G*, *S*, *V*, and the pairs (s,v) as generated by $G(1^n)$.

3. The Signature Scheme.

Our aim is to prove that under the assumption that there exist one-way permutations with trapdoor, there exist signature schemes secure against *CMA*. The scheme that we shall present is based on the work of Bellare and Micali [BM88] and we shall refer to it as *BM* scheme. However, the underlying ideas of this scheme appear already in the work by Goldwasser, Micali and Rivest [GMR], where a stronger complexity assumption is used.

3.1 General Description.

We saw in the previous lecture, that a scheme of the type of Rabin's [Ra78] signature scheme is not secure against *CMA*. An attacker could obtain enough information from several signatures of properly chosen messages to be able to break the system. The *BM* scheme solves this problem using two simple and ingenious ideas. We can state them informally as follows.

(i) Do not sign twice with the "same key".

(ii) Do not sign "directly" the document m chosen by the signee; sign only messages created by yourself.

Consider for example, Rabin's scheme where U publishes composite numbers n, product of two primes, as his public key, and keeps n's prime factorization as his secret key. The signature of a message m (a quadratic residue modulo n) is its square root modulo n. Verification is done by squaring modulo n. It is well known that this scheme is not secure against *CMA*: an enemy could ask for the signature of the square of a random number r^2 generated by himself, with the hope that this signature will correspond to a root of r^2 modulo n, different from the only roots that he knows, namely r and $-r$; this information would enable him to break the system. Therefore, it seems that if we could combine the two ideas above with Rabin's scheme, a secure system could be obtained.

Let us be more specific. Suppose that we had a signature system S_0, with the property that it is secure, provided it is used only once. We would like to use it to sign a message m_1. What we do, is to generate a new system S_1, and then, use the old system S_0 to sign both m_1 and S_1. By *system* we mean an instance of the signature scheme with specific keys s, v. Hence, by signing a system we mean signing these parameters (or parts of them while keeping the other parts as in the old system). When we want to sign another message, say m_2, we do it with the system S_1, and we prepare another system S_2, which will be signed with S_1, etc. In this way we are implementing principle (i), that is, signing messages created by others only once with the same system (i.e. the same pair of keys).

In terms of Rabin's scheme, the signature of m_1 is a square root of m_1 modulo n_0, where n_0 is the composite stored in the public file. Then, when U wants to sign a new message m_2, he randomly generates another composite n_1 (to generate the new system). The signature of m_2, is the square root of n_1 modulo n_0 (the signature of the new system with respect to the old one), *together* with the square root of m_2 modulo n_1. In general, the signature of m_t, consists of the whole sequence of square roots of n_{i+1} modulo n_i, $0 \leq i < t-1$, together with the square root of m_t modulo n_{t-1}.

With this scheme, the length of a signature grows linearly with the number of messages signed so far, because in order to validate the signature of a message m_{i+1}, the whole path $S_0, S_1, \ldots, S_i$ is needed; one needs to check that S_j was properly signed with S_{j-1}, for all $1 \leq j \leq i$, and then check that m_{i+1} was properly signed with S_i. Suppose that instead of creating one system S_j at the j–th step, we create two systems $S_{j,0}$ and $S_{j,1}$. As before, each one of these systems is used twice, once to sign a document *m*, and once to sign *two* new systems. In fact we are now using a binary tree, and thus, although the signature still includes the whole path from the root, the length of this path (and of the signature) is only logarithmic in the number of messages signed so far.

Another difficulty with the scheme as described above, is that *U* has to keep in memory the whole path (or tree in the case of the last improvement), in order to produce the next signature. One solution is to use a pseudo-random generator to produce the new systems (actually, to produce the coins tosses for the algorithm that generates the new systems), and remember the seed used initially. Then, when *U* wants to sign a new message, he can reproduce the sequence of systems used so far using the pseudo-random generator with the same seed, or alternatively remember the state of the pseudo-random generator.

Implementing idea (i) is not enough: consider Rabin's scheme in which giving to the enemy a single signature could be enough for an enemy to break the system. We need to implement idea (ii), also.

Denote the bits of *m* by $\sigma_1, \sigma_2, \ldots, \sigma_n$. The user *U* stores in a public file a vector consisting of $2n$ randomly chosen strings $\bar{\alpha} = \alpha_1^{(0)}, \alpha_1^{(1)}, \alpha_2^{(0)}, \alpha_2^{(1)}, \ldots, \alpha_n^{(0)}, \alpha_n^{(1)}$, and a string *y*. If we had a one way permutation f_y, with trapdoor $t(y)$, then a signature as required by (ii) would be:

$$f_y^{-1}(\alpha_1^{(\sigma_1)}),\ f_y^{-1}(\alpha_2^{(\sigma_2)}), \ldots,\ f_y^{-1}(\alpha_n^{(\sigma_n)}). \qquad (*)$$

These strings are the "signature" of *m* with respect to *y* and α, and will be denoted by $\tilde{S}_{y,\bar{\alpha}}(m)$.

Anybody can verify $\tilde{S}_{y,\bar{\alpha}}(m)$ is valid for *m*, by computing f_y (using the key *y*, stored in the public file) of the corresponding components of the vector $\bar{\alpha}$, and hence that it is indeed a signature of *m*, and not of something else. Moreover, nobody can forge a signature, since nobody can compute f_y^{-1} without having the trapdoor $t(y)$ (which is difficult to compute from *y*), and hence that it was *U* who signed and not somebody else.

Note that it is important to avoid signing more than once with the same *y* and $\bar{\alpha}$. For example, after seeing signatures for 0^n, and 1^n, with respect to *y* and $\bar{\alpha}$, one can sign any message w.r.t. *y* and $\bar{\alpha}$.

If for example, we implement *f* with Rabin's new system, we would be computing square roots of numbers generated by ourselves (the α's); an enemy can only decide of which of these numbers he wishes to obtain the square root: either of the first element, or of the second element of the pairs $\alpha_i^{(0)}, \alpha_i^{(1)}$, $1 \leq i \leq n$.

With these ideas in mind, let us proceed to describe formally the *BM* signature scheme, which together with its proof of security against *CMA* establishes the following theorem.

Theorem.

If there exist one-way permutations with trapdoor, then there exist signature schemes secure against *CMA*.

3.2 Definition of the *BM* Signature Scheme.

In this subsection we formally define the *BM* Signature scheme (G, S, V). First we define the tools which we assume are available to build the signature scheme: a collection of one-way permutations with trapdoor, with non-negligible domains. The following definition of a collection of one-way permutations with trapdoor appears in Lecture No. 3, without condition 0).

(a) Complexity Assumptions.

Assume there exists a collection of one-way permutations with trapdoor, with non-negligible domains,

$$\{f_y : D_y \to D_y\}_{y \in Y}$$

where Y is an infinite set (of indices), and D_y, for $y \in Y$ is a finite set. And the following conditions are satisfied:

0) The domains D_y have non-negligible size. Namely, there exists a constant d such that for every n and $y \in Y \cap \{0,1\}^n$: $|D_y| \geq 2^n \cdot n^{-d}$.

1) There exists a probabilistic polynomial-time algorithm A_1 which on input 1^n samples a pair (y, $t(y)$), where $y \in Y$, and $t(y)$ is the *trapdoor* for y.

2) There exists a probabilistic polynomial-time algorithm A_2 which for all $y \in Y$ samples elements out of D_y.

3) There exist a polynomial-time algorithm A_3 such that

$$\forall y,\ \forall x,\ A_3(y,\ x) = f_y(x)$$

4) There exist a polynomial-time algorithm A_4 such that

$$A_4(t(y),\ y,\ z) = f_y^{-1}(z)$$

5) For any probabilistic polynomial-time algorithm A^*, for any $c > 0$, and sufficiently large n,

$$Pr\left[A^*(f_y(x),\ y) = x\right] < \frac{1}{n^c}$$

where the probability is taken over the coin tosses of A^*, y chosen according to $A_1(1^n)$, and x chosen from D_y by A_2.

In the sequakl, we shall assume that the domains D_y are equal to $\{0,1\}^n$, for $|y| = n$. As we shall now see, there is no loss of generality as long as the domains are of non-negligible size.

Lemma: If there exists a collection of one-way permutations with trapdoor, with non-negligible domains D_y, then there there exists a collection of one-way permutations with trapdoor, such that for every y of length n, the set $D_y = \{0,1\}^n$.

Proof: Given a (collection of) one-way trapdoor permutation f_y on a subset D_y of $\{0, 1\}^n$, such that $|D_y| \geq 2^n \cdot n^{-d}$, (for some d) extend f_y to $\{0, 1\}^n$, by defining it to be the identity function on $\{0, 1\}^n - D$. This yields a weak one-way permutation $\overline{f}_y$ on $\{0, 1\}^n$.

One can use the cross product construction of [Y], described in Lecture #2, to obtain a strong one-way trapdoor permutation which fits our definition. In a few words, what one would do is to define a permutation F_y on

$$\{0, 1\}^n \times \cdots \times \{0, 1\}^n$$

by $F_y(x_1, ..., x_{n^{d+2}}) = (\overline{f}_y(x_1), ..., \overline{f}_y(x_{n^{d+2}}))$. This will be hard to invert on all but a negligible fraction of its domain. ■

(b) The Key Generator *G*.

The algorithm $G(1^n)$ produces the following strings:

$$(y,\ t(y)),\ \overline{\alpha},\ \overline{\beta}$$

where $G(1^n)$ produces $(y,\ t(y))$ using $A_1(1^n)$, and

$$\overline{\alpha} = \alpha_1^{(0)}, \alpha_1^{(1)}, \alpha_2^{(0)}, \alpha_2^{(1)}, \ldots, \alpha_n^{(0)}, \alpha_n^{(1)},$$

$$\overline{\beta} = \beta_1^{(0)}, \beta_1^{(1)}, \beta_2^{(0)}, \beta_2^{(1)}, \ldots, \beta_n^{(0)}, \beta_n^{(1)}$$

and the $\alpha_i^{(j)}$'s and $\beta_i^{(j)}$'s are chosen from $D_{|y|}$, using $A_2(|y|)$.

The *public key v* stored in the public file consists of y, $\overline{\alpha}$ and $\overline{\beta}$. The *secret key s* contains the strings of v, together with $t(y)$.

Conventions for Signing and Verification

The *message space M* is the set of all binary strings of n bits. The signer U receives requests for signatures of a sequence of messages, the ith, $i \geq 1$, message of the sequence is denoted by m_i.

The vector $\bar{\alpha}$ will be used to "sign" the messages m (as in (*)), while the vector $\bar{\beta}$ will be used to "sign" new systems, that is, new y's that allow the use of new permutations f_y's. The same vectors $\bar{\alpha}$ and $\bar{\beta}$ will be used in all the new systems created by U, thus only y's will be signed.

Let $y_0 = y$. Denote the bits of y_i by $\tau_{i,1}, \ldots, \tau_{i,n}$, and the bits of the message m_i by $\sigma_{i,1}, \ldots, \sigma_{i,n}$.

(c) The Signing Algorithm *S*.

The algorithm S uses as a building block an algorithm $\tilde{S}$, which has the role of "signing" a string $x = \sigma_1\sigma_2 \cdots \sigma_n$ with respect to a key y and a vector $\bar{\gamma}$, as in (*) :

$$\tilde{S}_{y,\bar{\gamma}}(x) = f_y^{-1}(\gamma_1^{(\sigma_1)}),\ f_y^{-1}(\gamma_2^{(\sigma_2)}), \ldots,\ f_y^{-1}(\gamma_n^{(\sigma_n)}).$$

Assume U has already signed t messages. To sign the message m_{t+1}, user U generates a number y_t using algorithm A_1 with input 1^n. The *signature* for m_{t+1} with respect to s consists of:

- The sequence of strings

$$y_1, y_2, \ldots, y_{t-1}$$

which were generated for the signatures of the previous messages, together with

- the new string

$$y_t$$

- For $0 \le i \le t-1$, a "signature" of y_{i+1} with respect to y_i and $\bar{\beta}$. Namely

$$\tilde{S}_{y_i,\bar{\beta}}(y_{i+1}) = f_{y_i}^{-1}(\beta_1^{(\tau_{i+1,1})}),\ f_{y_i}^{-1}(\beta_2^{(\tau_{i+1,2})}), \ldots,\ f_{y_i}^{-1}(\beta_n^{(\tau_{i+1,n})})$$

- A "signature" of m_{t+1} with respect to y_t, and $\bar{\alpha}$.

$$\tilde{S}_{y_t,\bar{\alpha}}(m_{t+1}) = f_{y_t}^{-1}(\alpha_1^{(\sigma_{t+1,1})}),\ f_{y_t}^{-1}(\alpha_2^{(\sigma_{t+1,2})}), \ldots,\ f_{y_t}^{-1}(\alpha_n^{(\sigma_{t+1,n})}).$$

Note that each y_i is used twice, one to "sign" y_{i+1} (w.r.t. $\bar{\beta}$), and one to sign m_{i+1} (w.r.t. $\bar{\alpha}$).

(d) The Verifying Algorithm *V*.

The procedure for verifying that a given signature is valid should be clear: Anybody can read the values $\bar{\alpha}$, $\bar{\beta}$ and y_0 stored in the public file, and check that for every $0 \le i \le t-1$, the string y_{i+1} has been "signed" with respect to y_i and $\bar{\beta}$, and then that m_{t+1} has been "signed" with respect to y_t and $\bar{\alpha}$.

3.3 Security of the *BM* Signature Scheme.

In this subsection we shall complete the proof of the Theorem. Namely, we shall prove that the *BM* signature scheme is secure against *CMA*. The proof is by contradiction; we assume that there exists an efficient forger F that has access to the signing algorithm S, and forges with respect to s. We show that this implies that by using F one can invert a one-way permutation f_y, a task assumed to be impossible for polynomial-time algorithms.

Specification of a Forger.

Assume for contradiction that F is a probabilistic polynomial-time algorithm that can forge signatures, that is, there exists $c > 0$, and an infinite sequence of n's such that

$$Pr\left(F^{S_s}(v)\ \ forges\right) > \frac{1}{n^c}.$$

We start by introducing some notation, and describing how does a forged signature looks like.

Suppose that F forges a signature for some message, say m. We know what the format of a signature for any message m looks like. It consists of a sequence of strings $y_0, y_1, \ldots, y_{t-1}$, together with their "signatures" $\tilde{S}_{y_i,\bar{\beta}}(y_{i+1})$, for $0 \le i < t-1$, and a "signature" of m, $\tilde{S}_{y_{t-1},\bar{\alpha}}(m)$, where y_0, $\bar{\alpha}$ and $\bar{\beta}$ appear under U's name in the public file.

Therefore, whatever string F claims is a signature should have this structure. So denote by

$$\tilde{y}_0, \tilde{y}_1, \ldots, \tilde{y}_t \ (= \tilde{m}_t)$$

the strings appearing in F's forged signature to $m = \tilde{m}_t$. where $\tilde{y}_t = \tilde{m}_t$ (we take the liberty of identifying $\tilde{y}_t$ with $\tilde{m}_t$). Thus, F should be able to present appropriate "signatures" of $\tilde{y}_{j+1}$ with respect to $\tilde{y}_j$, for $0 \le j < t$. Also note that $\tilde{y}_0 = y_0$.

Moreover, F forged the signature, which means that it is different from the signatures F has obtained via the chosen message attack. Let $y_0, y_1, \ldots, y_t$, where $y_t = m_t$ be the sequence used in the t-th signature obtained by the chosen message attack. (Note that y_0 is in U's public file, the other y_i's, $1 \le i < t$, are chosen by the signer while only m_t is chosen by F). Recall, $\tilde{y}_t \ne y_t$. Hence, there is a location in the sequence where F "forges" for the first time, i.e. there is a q, $1 \le q \le t$, such that $y_i = \tilde{y}_i$ for $i < q$, and $y_q \ne \tilde{y}_q$. Therefore, F was able to create the "signature" of $\tilde{y}_q$ with respect to y_{q-1}.

Now, before presenting the algorithm which inverts a one-way permutation, some explanations about the random choices that are going to be made.

Assume, without loss of generality, that algorithm F always asks for the signatures of l messages, and forges a signature for some message $\tilde{m}_t$, $t \le l$. The algorithm which we shall describe guesses the "place" where F forges; namely, it guesses three numbers q, j, and k specified in the sequal.

If F is able to forge with non-negligible probability, it is able to forge for the first time, with non-negligible probability, at a certain step q. This is because l is polynomial in n, and a non-negligible probability divided by a polynomial remains non-negligible. Therefore, we can choose at random q.

There are two possibilities for F, either

$q = t$: forge directly a message, i.e. "sign" $\tilde{m}_q$ w.r.t. y_{q-1}.

or

$q < t$: forge a "signature" of a system $\tilde{y}_q$ w.r.t. y_{q-1}.

We assume $q < l$; the proof for the other case is similar.

We have then,

$$\tilde{y}_{q-1} = y_{q-1}$$

$$\tilde{y}_q \neq y_q$$

and thus, instead of the "signature"

$$\tilde{S}_{y_{q-1},\bar{\beta}}(y_q)$$

given by algorithm S, there is the forged "signature" which F managed to produce with non-negligible probability

$$\tilde{S}_{y_{q-1},\bar{\beta}}(\tilde{y}_q).$$

Since the strings y_q and $\tilde{y}_q$ are different, they are different at some bit j. Again, if we choose a random j, then there is a non-negligible probability that the strings y_q and $\tilde{y}_q$ differ at the jth bit. This is because they differ at *some* bit with non-negligible probability, and because there are only n possible values for j.

Finally, we choose k randomly from $\{ 0,1 \}$. If $k = 0$, we assume that $\tau_{q,j} = 0$ while in the string $\tilde{y}_q$, the bit $\tilde{\tau}_{q,j} = 1$; otherwise we assume that $\tau_{q,j} = 1$, and $\tilde{\tau}_{q,j} = 0$. One of these possibilities holds with non-negligible probability, hence, we may assume that the first one holds.

An Algorithm to Invert the One-way Permutation.

We now present an algorithm F^*, which using F is able to invert the one-way permutation f_y (a contradiction to property 5) in section 3.2); namely, its input and output are :

Input: y, $f_y(x)$

Output: x (with non-negligible probability)

Namely, there exist a constant $c > 0$, and an infinite sequence $n_1, n_2, \ldots$ and for every i

$$Pr\left(F^*(f_y(x)) = x\right) \geq \frac{1}{n_i{}^c},$$

where the probability space is defined over the inputs x of length n_i with uniform distribution, the strings y as produced by $A_1(1^{n_i})$, the vectors $\bar{\alpha}$ and $\bar{\beta}$ as produced by A_2, the sequences of inner coin tosses of F^* for inputs of length n_i, with uniform distribution.

Algorithm F^* (input: y, $f_y(x)$. Output: x.)

```
(0)  Randomly choose $q$, $j$ and $k$, $1 \le q \le l$, $1 \le j \le n$, $0 \le k \le 1$.   /* Expect $F$ to forge at step $q$, and bit $j$. */
(1)  Choose two vectors $\bar{\gamma}_1$ and $\bar{\gamma}_2$ each with $2n$ entries randomly chosen from $D_n$. Let $\bar{\alpha} = f_y(\bar{\gamma}_1)$ and
     $\bar{\beta} = f_y(\bar{\gamma}_2)$.
(2)  Replace $\beta_j^{(1)}$ by $f_y(x)$.   /* replace $\beta_j^{(k)}$ by $f_y(x)$, w.l.o.g. let $k = 1$ */
(3)  Choose $l$ pairs $(y_0, t(y_0)), (y_1, t(y_1)), \ldots, (y_{l-1}, (t(y_{l-1}))$ , using algorithm $A_1(1^n)$. Replace $y_{q-1}$ by $y$;
     assume that the $j$th bit of $y_q$ is 0,    otherwise start all over.
(4)  Let $F$ ask for signatures of $l$ messages $m_1, \ldots, m_l$, reply to $F$ the corresponding signatures.
         /* this can be done as shown in Claim 1   */
(5)  Take the output of $F$, for step $q$ :   $\tilde{S}_{y_{q-1}, \bar{\beta}}(\tilde{y}_q)$,
     Output the $j$th component of $\tilde{S}_{y_{q-1}, \bar{\beta}}(\tilde{y}_q)$     /* which is equal to   $f_{y_{q-1}}^{-1}(\beta_j^{(1)})$ with non-negligible proba-
     bility */.
end
```

To complete the proof of the Theorem, we shall prove the following two claims.

Claim 1: Algorithm F^* is able to compute the signatures at step (4).

Proof: We show that F^* is able to compute the signatures required by F. For $i \ne q-1$, it can compute both,

$$\tilde{S}_{y_i, \bar{\alpha}}(m_{i+1})$$

and

$$\tilde{S}_{y_i, \bar{\beta}}(y_{i+1})$$

because in step (3) the algorithm F^* generated y_i together with the trapdoors $t(y_i)$, for $i \ne q-1$.

For $i = q-1$, algorithm F^* has uses y in role of y_i. Although F^* does not "know" $t(y)$ it "knows" the value of f_y^{-1} all all α's and all β's (except the input string replacing $\beta_j^{(1)}$), since F^* has produced these α's and β's by selecting some γ's and applying f_y to them! It follows that F^* can compute

$$\tilde{S}_{y, \bar{\alpha}}(m_q)$$

since it is not more than a subset (defined by m_q) of the components of $\bar{\gamma}_1$. Algorithm F^* can also compute

$$\tilde{S}_{y,\,\bar{\beta}}(y_q)$$

because what it has to compute is a subset (defined by y_q) of the components of $\bar{\gamma}_2$. Note that this subset does **not** include the component corresponding to $\beta_j^{(1)}$, which is the only place where F^* would have trouble. That is, F^* is not asked to compute $f_y^{-1}\, f_y(x)$. ■

Claim 2: The output of F^* is x, with non-negligible probability.

Proof: Provided that q, j and k were guessed correctly, with non-negligible probability, the output of the algorithm is indeed x : The jth component of $\tilde{S}_{y_{q-1},\,\bar{\beta}}(\tilde{y}_q)$ is $f_{y_{q-1}}^{-1}(\beta_j^{(1)})$ since we are assuming that $\tilde{\tau}_{q,\,j} = 1$.

Also, $\beta_j^{(1)} = f_y(x)$, and $y_{q-1} = y$, and therefore

$$f_{y_{q-1}}^{-1}(\beta_j^{(1)}) \;=\; f_y^{-1}(f_y(x)).$$

Namely, the output is correct provided that the jth bit of y_q is 0, and F forges the "signature" for $\tilde{y}_q$, s.t. the jth bit of $\tilde{y}_q$ is 1. Our assumption is that these conditions hold with non-negligible probability, when the public-key is chosen by the key generator and the other y_i's are selected by the signing algorithm. We stress that the α's, β's, and y_i's chosen by F^* together with the input y have the same probability distribution! It follows that F^* inverts f_y with non-negligible probability, a contradiction. ■