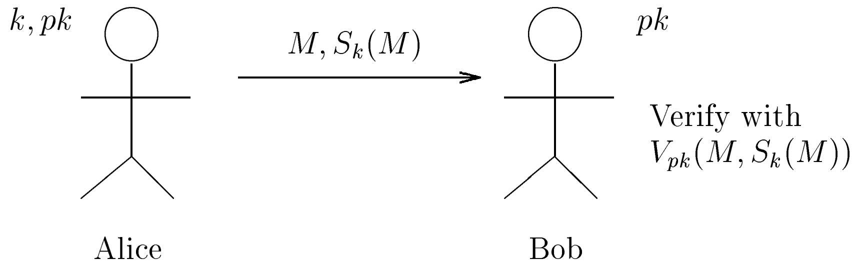
03/19/02 - Public Key Signature Schemes

1 Motivation and Definition



The goal of a public key signature scheme is to provide integrity of sent messages. If Alice sends Bob a message, Bob would like to know that the message he receives came from Alice and that it is the message that Alice actually sent. In a signature scheme Alice uses a secret key to compute a signature for her message and sends this signature along with her message. Bob verifies that the signature matches the message and thus gains some assurance that the message he has is one that Alice intended to send him.

The idea is similar to that of a MAC in that its goal is to ensure integrity, but different in that Alice and Bob need not share a secret key. Instead of sharing a s secret key, as in a MAC, Alice and Bob must both have a public/secret key pair.

The components of a public key signature scheme are

- A signing algorithm, $S : K \times \{0,1\}^* \to Y$ which takes a secret key and a message and outputs a signature for that message.
- A verification algorithm $V : PK \times \{0,1\}^* \times Y \to \{0,1\}$ which takes a public key, a message and a signature and verifies that the signature is valid for that message.
- A key generation algorithm G : randomness $\to PK \times K$ which generates public/secret key pairs.

Suppose Alice sends Bob a message signed with her secret key that says she'll buy his house for \$1M but later changes her mind, Bob can show an impartial judge the message that Alice signed. Assuming her signature scheme is secure, Alice is the only one who could have signed the message. So Bob can prove that Alice said she'll buy his house for \$1M. This property of public key signature schemes is known as *non-repudiation.* Notice that non-repudiation arises from the asymmetric nature of public key signature schemes - MACs clearly do not have this property since either Alice or Bob could sign the message. Formally stating what constitutes a signature scheme that guarantees non-repudiation is tricky and outside the scope of this lecture.

Intuitively, a signature scheme should be both

- correct: If there's no adversary messing things up, Bob should always accept Alice's messages: $\forall M, (pk, k) \leftarrow G : V_{pk}(M, S_k(M)) = 1$. If V is a randomized algorithm, then V must accept valid messages with high probability.

- secure: Even if an adversary can see signatures for a few chosen messages, she cannot generate a valid signature for a new message.

DEFINITION 1 *A public key signature scheme is* (t, q, ε)-secure against existential forgery attacks *if for all algorithms A^{S_k} furnished with a signing oracle S_k, running in time less than t, making at most q queries to its oracle and outputting a message, signature pair on which it did not query the oracle,*

$$\mathbf{Pr}_{(pk,k)\leftarrow G}[V_{pk}(m', y') = 1 \text{ where } (m', y') \leftarrow A^{S_k}(pk)] \leq \varepsilon$$

2 Example Signature Schemes

For the rest of this lecture we consider five signature schemes based around RSA. In describing them we'll use N for the modulus, (e, N) for the public key and d for the secret key and M for the message, interpreted as a $n = \lceil \log_2 N \rceil$-bit number between 0 and N. In the next section we'll discuss the security of the latter four.

EXAMPLE 1 (PLAIN RSA) Just use a trapdoor permutation (in this case, RSA) directly to sign the message: $S(M) = M^d \pmod N$. This method is insecure. The following attack uses two chosen message queries to forge a signature for an arbitrary message, M. Find $S_d(2) = 2^d \pmod N$ and $S_d(2M) = 2^d \cdot M^d = 2^d S_d(M) \pmod N$. Then divide to obtain $S_d(M)$. This attack works essentially because RSA used in this way has the unfortunate property that $S(M_1 \cdot M_2) = S(M_1) \cdot S(M_2)$

A less powerful attack produces a valid message/signature pair with no chosen message queries (but the message may not be particularly sensible). In particular for any $x \in (\mathbb{Z}/n\mathbb{Z})^*$, x is a valid signature for the message $x^e \pmod N$. This attack works no matter which trapdoor permutation we use.

These attacks suggest a more clever construction using hash functions.

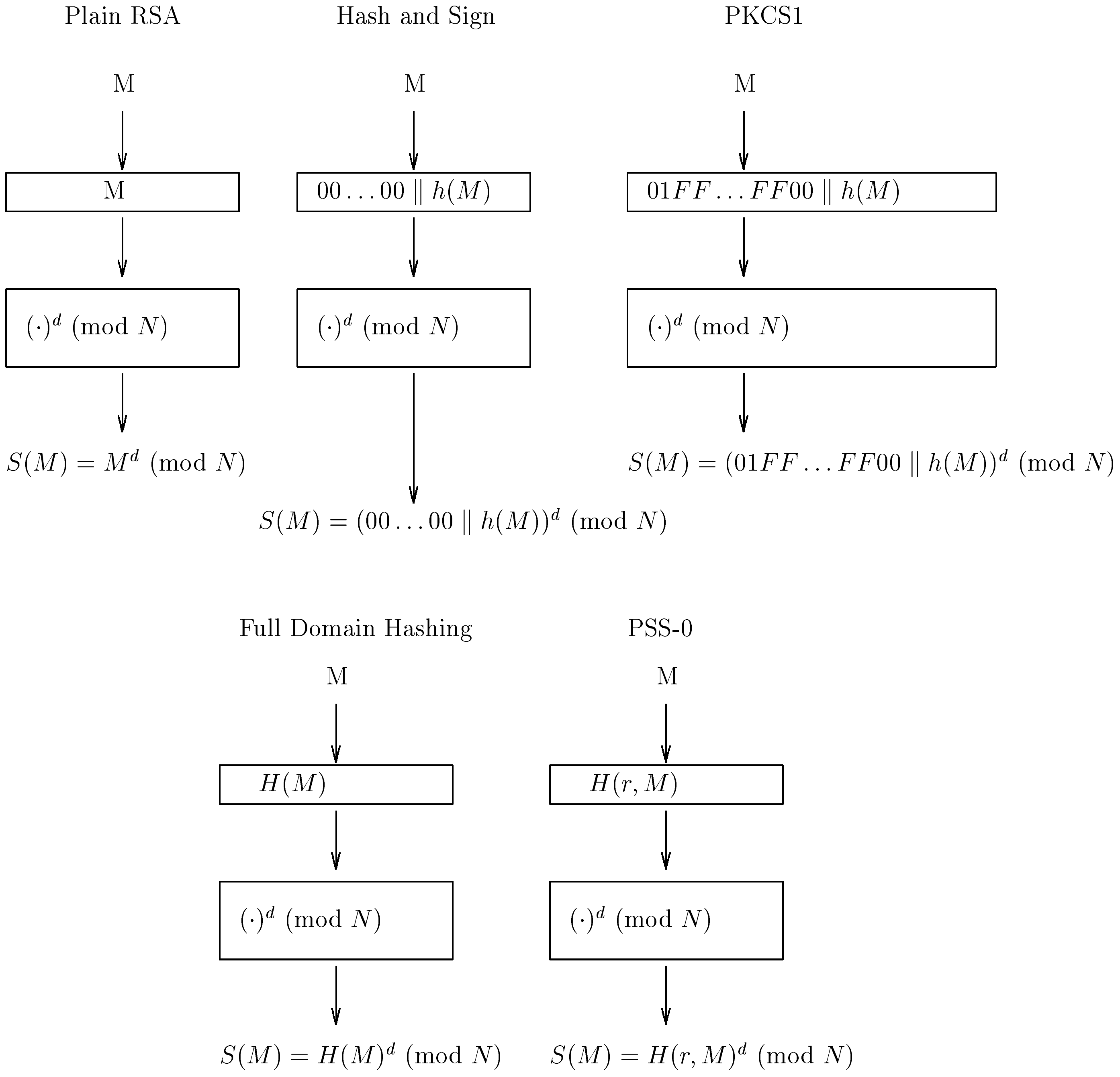
EXAMPLE 2 (HASH AND SIGN) Hash the message, pad the high order bits of the output of the hash function with enough zeros for it to be n bits long, i.e. $S(M) = (00\ldots00 \parallel h(M))^d \pmod N$ where h is some hash function. Note that h has no secret key and is thus publically computable.

Not just any hash function is sufficient. For a hash function to protect against the attacks on Plain RSA it should

- be one-way, otherwise a simple variant of the second attack still works, i.e. $S(h^{-1}(x^e)) = (h(h^{-1}(x^e)))^d = x \pmod N$.
- resist collisions i.e. it should be hard to get M_1, M_2 such that $h(M_1) = h(M_2)$. It would be very bad, for instance, if the messages "I'll buy your house for \$1M" and "I'll buy your house for \$3.5M were to collide.
- destroy algebraic structure e.g. $h(M_1 \cdot M_2) \neq h(M_1) \cdot h(M_2)$ or else the first attack would still work.

These conditions are certainly all necessary, but whether they are sufficient is unknown.

Figure 1: Five RSA Based Signature Schemes



Example 3 (PKCS1) This signature scheme is like the previous one, except instead of padding with zeros, we pad with the bytes `01 FF ··· FF 00` i.e. $S(M) = (01FF \cdots FF00 \parallel h(M))^d \pmod{N}$.

Example 4 (Full Domain Hashing) Apply a hash function which maps inputs to n-bit outputs. $S(M) = H(M)^d \pmod{N}$.

Example 5 (PSS-0) For each message, apply a hash function chosen randomly from a family of hash functions that map inputs to n-bit outputs. Notice that we must also send the index of the hash function. $S(M) = r, H(r, M)^d \pmod{N}$

3 Security of Example Signature Schemes

In order to consider the security of these signature schemes on their own, we'll assume that RSA is a secure trapdoor permutation and that we have a random oracle available for use as a hash function. In this scenario, any problems with the signature scheme as a whole are clearly problems of the scheme itself rather than weaknesses of its building blocks.

- Even in this model, Hash and Sign is insecure. Padding with zeros seems suspect. The guarantee of a trapdoor permutation is that it is hard to invert on random input, but padding with zeros means that the input will be concentrated in the low numbers. It turns out that RSA is easier to invert for such inputs and thus the Hash and Sign signature scheme is insecure.
- PKCS1 seems suspect for the same reasons, but no attack on it is known. No proof of security is known either.
- Full Domain Hashing is secure. We present a proof below.
- PSS-0 is also secure. In the next lecture a stronger security result than the one for FDH is proven about it.

4 Full Domain Hashing is Secure

We'd like to show that FDH ($S(M) = H(M)^d \pmod{N}$) is a secure public key signature scheme. As usual, we proceed by contradiction, assuming that we have an adversary that could make an existential forgery attack and show that we can use this adversary to break RSA. First, we consider the easy case where the adversary makes no queries to the signature oracle.

Lemma 1
If RSA is a (t, ε)-secure trapdoor permutation, then FDH is a $(t, 0, \varepsilon)$-secure public key signature scheme in the random oracle model.

Suppose, by way of contradiction, that $A^{R,-}$ is an adversary that runs in time less than t, makes no queries to its signing oracle, has access to random oracle R and outputs a valid signature/message pair with probability greater than ε. We'll construct an adversary that inverts RSA, i.e. given some z and the public key e, it returns $z^d \pmod{N}$ with probability greater than ε.

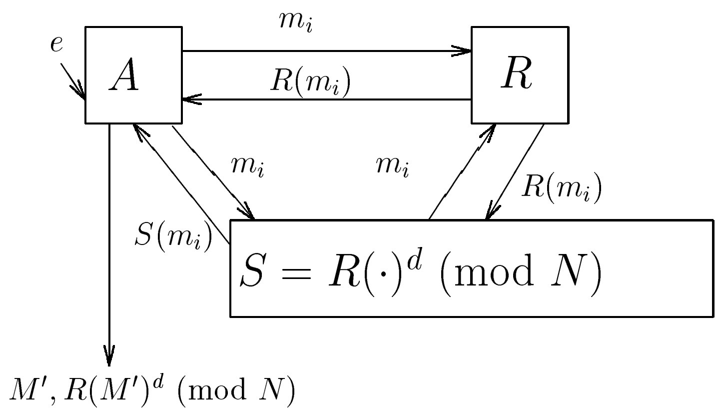


Figure 2: Anatomy of A, an adversary to break FDH

So A gets to query the random oracle, R, perform some computation and output an answer $(M, R(M)^d \pmod N)$. Without too much loss of generality, A outputs the result of one of its queries of R raised to the d mod N. Since a query to R takes one unit of time, A can query R at most t times.

- Idea 1: Replace A's random oracle R with an oracle, $\hat{R}$ that picks j at random, independent of everything else, and outputs z as the result of A's j^{th} query and otherwise functions like R. The results from $\hat{R}$ would be distributed uniformly, so A would function properly. Since j is chosen independently, A has no idea which query was rigged and so it outputs $z^d \pmod N$ with probability at least $\frac{\varepsilon}{t}$. This foreshadows what we'll have to do to prove the theorem (and where that factor of t comes from), but for this lemma, we can do better.
- Idea 2: We'll use the multiplicative structure of RSA to our advantage. For query m_i, $\hat{R}$ will pick r_i uniformly, independent of everything else at random and return $r_i^e z$. Then A will somehow pick one of the query results and raise it to the d mod N yielding $(r_i^e z)^d = r_i z^d \pmod N$ and output $(m_i, r_i z^d)$. So we can divide by r_i to find z^d. This method succeeds whenever A succeeds in forging a signature and thus proves the lemma.

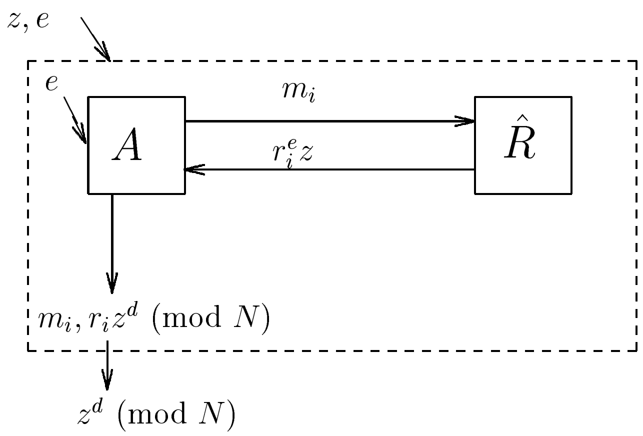


Figure 3: Using A to invert RSA when A makes no signing queries

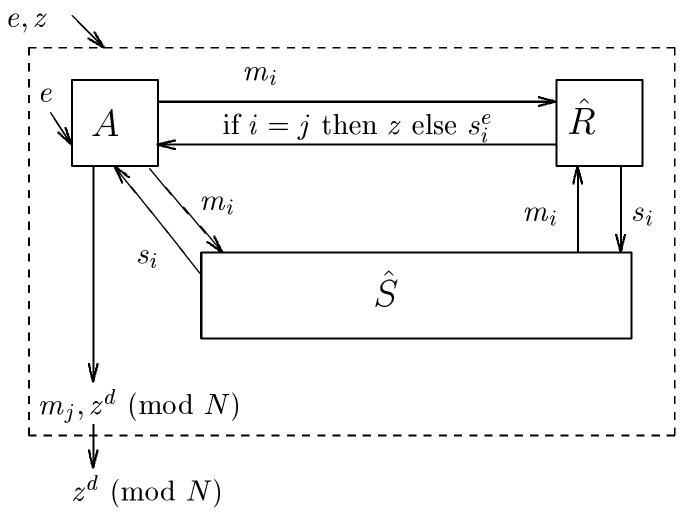


Figure 4: Using A to invert RSA when A does make signing queries

THEOREM 2
If RSA is a (t, ε)-secure trapdoor permutation, then FDH is a $(t, q, t\varepsilon)$-secure public key signature scheme in the random oracle model.

Notice that this theorem only shows that FDH is $(t, q, t\varepsilon)$-secure. The factor of t makes this a fairly weak result. There is actually a paper which shows that FDH is $(t, q, q\varepsilon)$-secure which is a much stronger result since typically $q << t$. This result is not tight - as far as Prof. Wagner knows, the best known attack only breaks FDH with probability ε.

Again we use assume by way of contradiction that we have some adversary $A^{R,S}$ that runs in time less than t, makes q queries to its signing oracle S, has access to random oracle R and outputs a valid signature/message pair with probability greater than $t\varepsilon$. We'll construct an adversary that runs A with a "signing oracle" $\hat{S}$ and "random oracle" $\hat{R}$ such that we can invert RSA with probability greater than ε, i.e. given some z and the public key e, it returns $z^d \pmod N$ with probability $\frac{1}{t}$ conditioned on $A^{\hat{R},\hat{S}}$ succeeding.

First of all, since we don't know d, we can't actually sign messages, so when we run A, we have to give it a signing oracle $\hat{S}$ that just outputs uniform random numbers independent of everything else. So for a new query m_i $\hat{S}(m_i) = s_i$ where s_i is picked randomly, independent of everything else. If m_i is a message that either $\hat{S}$ or $\hat{R}$ has seen before, return something consistent.

It would be nice to use idea 2 from the proof of the lemma to construct $\hat{R}$, but $\hat{R}(m_i) = r_i^e z \Leftrightarrow \hat{S}(m_i) = r_i z^d$ which we cannot produce. So, the difficult part is assuring that the output of our random oracle, $\hat{R}$, is consistent with the output of $\hat{S}$ in the case where A queries both on the same message. If $\hat{R}(m_i)$ just picked s_i at random and returned s_i^e it would ensure that it could later sign m_i. $\hat{S}(m_i) = s_i \Leftrightarrow \hat{R}(m_i) = s_i^e$. Unfortunately if A chose to raise one of these outputs to the d, we'd gain no useful information.

Remember that there is one output from $\hat{R}$ that A is not allowed to ask for a signature on - i.e. the one for which it is forging a signature. So $\hat{R}$ will pick a random j independent of everything else. On query j it will output z otherwise it will output s_i^e for random s_i. The distribution of responses form $\hat{R}$ is uniform. Since A does not know j (j was chosen independently of everything else), it will chose its output in whatever way it normally would and we succeed in computing z^d exactly when j is the random oracle query that A raises to the d, i.e. with probability at least $\frac{1}{t}$. QED.