CPSC 467b: Cryptography and Computer Security

Lecture 15

Michael J. Fischer

Department of Computer Science
Yale University

March 3, 2010

Digital Signatures (continued)

Desired security properties of digital signatures

Digital signatures must be difficult to forge.

Some increasingly stringent notions of forgery-resistance:

- Resistance to forging valid signature for particular message m.
- Above, but where adversary knows a set of valid signed messages $(m_1, s_1), \ldots, (m_k, s_k)$, and $m \notin \{m_1, \ldots, m_k\}$.
- Above, but where adversary can choose a set of valid signed messages, specifying either the messages (corresponding to a chosen plaintext attack) or the signatures (corresponding to chosen ciphertext attack).
- Any of the above, but where one wishes to protect against generating any valid signed message (m', s') different from those already seen, not just for a particular predetermined m.

Forging random signed messages

RSA signatures are indeed vulnerable to forgery of random signed messages.

An attacker simply generates chooses s' at random and computes $m' = E_e(s')$.

The signed message (m', s') is trivially valid since the verification predicate is simply $m' = E_e(s')$.

Importance of random signed messages

One often wants to sign random strings.
The ability of an attacker to generate valid random signed messages is a real drawback for certain practical applications.

For example, in the Diffie-Hellman key exchange protocol discussed in Lecture 12, Alice and Bob exchange random-looking numbers $a = g^x \bmod p$ and $b = g^y \bmod p$.

In order to discourage man-in-the-middle attacks, they may wish to sign these strings. (This assumes that they already have each other's public signature verification keys.)

However, Mallory could feed bogus signed values a' and b' to Bob and Alice, respectively. The signatures would check, and both would think they had successfully established a shared key k when in fact they had not.

Adding redundancy

One way to defeat the adversary's ability to generate valid random signed messages is to put redundancy into the message, for example, by prefixing a fixed string σ to the front of each message before signing it.

Instead of taking $S_d(m) = D_d(m)$, one could take

$$S_d(m) = D_d(\sigma m).$$

The corresponding verification predicate would then be

$$V_e(m, s) \Leftrightarrow \sigma m = E_e(s).$$

Security of signatures with fixed redundancy

The security of this scheme depends on the mixing properties of the encryption and decryption functions, that is, that each output bit depends on all of the input bits.

Not all cryptosystems have this mixing property.

For example, a block cipher used in ECB mode (see lectures 3 and 5) encrypts a block at a time, so each block of output bits depends only on the corresponding block of input bits.

Forging signatures

Suppose it happens that

$$S_d(m) = D_d(\sigma m) = D_d(\sigma) \cdot D_d(m).$$

Then Mallory can forge random messages assuming he knows just one valid signed message (m_0, s_0). Here's how.

- He knows that $s_0 = D_d(\sigma) \cdot D_d(m)$, so from s_0 he extracts the prefix $s_{00} = D_d(\sigma)$.
- He now chooses a random s'_{01} and computes $m' = E_e(s'_{01})$ and $s' = s_{00} \cdot s'_{01}$.
- The signed message (m', s') is valid since $E_e(s') = E_e(s_{00} \cdot s'_{01}) = E_e(s_{00}) \cdot E_e(s'_{01}) = \sigma m'$.

Signing message digests

A better way to prevent forgery is to sign a *message digest* of the message rather than sign m itself.

A message digest function h, also called a *cryptographic one-way hash function* or a *fingerprint function*, maps long strings to short random-looking strings.

- To sign a message m, Alice computes $S_d(m) = D_d(h(m))$.
- To verify the signature s, Bob checks that $h(m) = E_e(s)$.

Forging signed message digests

For Mallory to generate a forged signed message (m', s') he must somehow come up with m' and s' satisfying

$$h(m') = E_e(s') \tag{1}$$

That is, he must find m' and s' that both map to the same string, where m' is mapped by h and s' by E_e.

Two natural approaches for attempting to satisfying (1):

1. Pick m' at random and solve for s'.
2. Pick s' at random and solve for m'.

Solving for s'

Approach 1:

$$h(m') = E_e(s') \tag{1}$$

To solve for s' given m' requires computing

$$E_e^{-1}(h(m')) = D_d(h(m')) = s'.$$

Alice can compute D_d, which is what enables her to sign messages.

But Mallory presumably cannot compute D_d without knowing d, so this approach doesn't work.

Solving for m'

Approach 2:

$$h(m') = E_e(s') \tag{1}$$

To solve for m' given s' requires "inverting" h.

Since h is many-one, a value $y = E_e(s')$ can have many "inverses" or *preimages*.

To successfully forge a signed message, Mallory needs to find only one value m' such that $h(m') = E_e(s')$.

However, the defining property of a cryptographic hash function is that, given y, it should be hard to find any $x \in h^{-1}(y)$.

Hence, Mallory cannot feasibly find m' satisfying 1.

Other attempts

Of course, these are not the only two approaches that Mallory might take.

Perhaps there are ways of generating valid signed messages (m', s') where m' and s' are generated together.

I do not know of such a method, but this doesn't say one doesn't exist.

More advantages of signing message digests

Another advantage of signing message digests rather than signing messages directly: the signatures are shorter.

An RSA signature of m is roughly the same length as m.

An RSA signature of $h(m)$ is a fixed length, regardless of how long m is.

For both reasons of security and efficiency, signed message digests are what is used in practice.

We'll talk more about message digests later on.

What does a digital signature imply?

We like to think of a digital signature as a digital analog to a conventional signature.

- A conventional signature binds a person to a document. Barring forgery, a valid signature indicates that a particular individual performed the action of signing the document.
- A digital signature binds a signing key to a document. Barring forgery, a valid digital signature indicates that a particular signing key was used to sign the document.

However, there is an important difference. A digital signature only binds the signing key to the document.

Other considerations must be used to bind the individual to the signing key.

Disavowal

An individual can always disavow a signature on the grounds that the private signing key has become compromised.

Here are two ways that this can happen.

- Her signing key might be copied, perhaps by keystroke monitors or other forms of spyware that might have infected her computer, or a stick memory or laptop containing the key might be stolen.
- She might deliberately publish her signing key for the express purpose of relinquishing responsibility for documents signed by it.

For both of these reasons, one cannot conclude without a reasonable doubt that a digitally signed document was indeed signed by the purported holder of the signing key.

Practical usefulness of digital signatures

This isn't to say that digital signatures aren't useful; only that they have significantly different properties than conventional signatures.

In particular, they are subject to disavowal by the signer in a way that conventional signatures are not.

Nevertheless, they are still very useful in situations where disavowal is not a problem.

Message Digest Functions

Message digest function

Let $\mathcal{M}$ be a message space and $\mathcal{H}$ a hash value space, and assume $|\mathcal{M}| \gg |\mathcal{H}|$.

A *message digest* (or *cryptographic one-way hash* or *fingerprint*) function h maps $\mathcal{M} \to \mathcal{H}$.

A *collision* is a pair of messages m_1, m_2 such that $h(m_1) = h(m_2)$, and we say that m_1 and m_2 *collide*.

Because $|\mathcal{M}| \gg |\mathcal{H}|$, h is very far from being one-to-one, and there are many colliding pairs. Nevertheless, it should be hard for an adversary to find collisions.

Collision-avoidance properties

We consider three increasingly strong versions of what it means to be hard to find collisions:

One-way property: Given $y \in \mathcal{H}$, it is hard to find $m \in \mathcal{M}$ such that $h(m) = y$.

Weak collision-free property: Given $m \in \mathcal{M}$, it is hard to find $m' \in \mathcal{M}$ such that $m' \neq m$ and $h(m') = h(m)$.

Strong collision-free property: It is hard to find colliding pairs (m, m'). (What does this mean in a complexity-theoretic sense?)

These definitions are rather vague, for they ignore issues of what we mean by "hard" and "find".

What does "hard" mean?

Intuitively, "hard" means that Mallory cannot carry out the computation in a feasible amount of time on a realistic computer.

What does "find" mean?

The term "find" may mean

- "always produces a correct answer", or
- "produces a correct answer with high probability", or
- "produces a correct answer on a significant number of possible inputs with non-negligible probability".

The latter notion of "find" says that Mallory every now and then can break the system. For any given application, there is a maximum acceptable rate of error, and we must be sure that our cryptographic system meets that requirement.

One-way function

What does it mean for h to be one-way?

Intuitively, this means that no probabilistic polynomial time algorithm $A_h(y)$ produces a pre-image m of y under h with more than negligible probability of success.

This is only required for random y chosen according to a particular hash value distribution. There might be particular values of y on which A_h has non-negligible success probability.

The hash value distribution we have in mind is the one induced by h applied to uniformly distributed $m \in \mathcal{M}$.

The probability of y is proportional to $|h^{-1}(y)|$.

This means that h can be considered one-way even though algorithms do exist that succeed on low-probability subsets of $\mathcal{H}$.

Constructing one hash function from another

The following example might help clarify these ideas.

Let $h(m)$ be a cryptographic hash function that produces hash values of length n. Define a new hash function $H(m)$ as follows:

$$H(m) = \begin{cases} 0 \cdot m & \text{if } |m| = n \\ 1 \cdot h(m) & \text{otherwise.} \end{cases}$$

Thus, H produces hash values of length $n + 1$.

- $H(m)$ is clearly collision-free since the only possible collisions are for m's of lengths different from n.
- Any colliding pair (m, m') for H is also a colliding pair for h.
- Since h is collision-free, then so is H.

H is one-way

Not so obvious is that H is one-way.

This is true, *even though H can be inverted for 1/2 of all possible hash values y*, namely, those that begin with 0.

The reason this doesn't violate the definition of one-wayness is that only 2^n values of m map to hash values that begin with 0, and all the rest map to values that begin with 1.

Since we are assuming $|\mathcal{M}| \gg |\mathcal{H}|$, the probability that a uniformly sampled $m \in \mathcal{M}$ has length exactly n is small.

Strong implies weak collision-free

There are some obvious relationships between properties of hash functions that can be made precise once the underlying definitions are made similarly precise.

Fact

If h is strong collision-free, then h is weak collision-free.

Proof that strong ⇒ weak collision-free

Proof (Sketch).

Suppose h is *not* weak collision-free. We show that it is not strong collision-free by showing how to enumerate colliding message pairs.

The method is straightforward:

- Pick a random message $m \in \mathcal{M}$.
- Try to find a colliding message m'.
- If we succeed, then output the colliding pair (m, m').
- If not, try again with another randomly-chosen message.

Since h is not weak collision-free, we will succeed on a significant number of the messages, so we will succeed in generating a succession of colliding pairs. □

Speed of finding colliding pairs

How fast the pairs are enumerated depends on how often the algorithm succeeds and how fast it is.

These parameters in turn may depend on how large $\mathcal{M}$ is relative to $\mathcal{H}$.

It is always possible that h is one-to-one on some subset U of elements in $\mathcal{M}$, so it is not necessarily true that every message has a colliding partner.

However, an easy counting argument shows that U has size at most $|\mathcal{H}| - 1$.

Since we assume $|\mathcal{M}| \gg |\mathcal{H}|$, the probability that a randomly-chosen message from $\mathcal{M}$ lies in U is correspondingly small.

Strong implies one-way

In a similar vein, we argue that strong collision-free implies one-way.

Fact

If h is strong collision-free, then h is one-way.

Proof that string *Rightarrow* one-way

Proof (Sketch).

Suppose h is *not* one-way. Then there is an algorithm $A(y)$ for finding m such that $h(m) = y$, and $A(y)$ succeeds with significant probability when y is chosen randomly with probability proportional to the size of its preimage. Assume that $A(y)$ returns $\perp$ to indicate failure.

The following randomized algorithm enumerates a sequence of colliding pairs:

1. *Choose random* m.
2. *Compute* $y = h(m)$.
3. *Compute* $m' = A(y)$.
4. *If* $m' \notin \{\perp, m\}$ *then output* (m, m').
5. *Start over at step 1.*

Proof (cont.)

Proof (continued).

Each iteration of this algorithm succeeds with significant probability ε that is the product of the probability that $A(y)$ succeeds on y and the probability that $m' \neq m$.

The latter probability is at least $1/2$ except for those values m which lie in the set of U of messages on which h is one-to-one (defined in the previous proof).

Thus, assuming $|\mathcal{M}| \gg |\mathcal{H}|$, the algorithm outputs each colliding pair in expected number of iterations that is only slightly larger than $1/\varepsilon$. □

Weak implies one-way

These same ideas can be used to show that weak collision-free implies one-way, but now one has to be more careful with the precise definitions.

Fact

If h is weak collision-free, then h is one-way.

Proof that weak ⇒ one-way

Proof (Sketch).

Suppose as before that h is *not* one-way, so there is an algorithm $A(y)$ for finding m such that $h(m) = y$, and $A(y)$ succeeds with significant probability when y is chosen randomly with probability proportional to the size of its preimage.

Assume that $A(y)$ returns $\perp$ to indicate failure. We want to show this implies that the weak collision-free property does not hold, that is, there is an algorithm that, for a significant number of $m \in \mathcal{M}$, succeeds with non-negligible probability in finding a colliding m'.

Proof (cont.)

Proof (continued).

We claim the following algorithm works:

Given input m:

1. *Compute $y = h(m)$.*
2. *Compute $m' = A(y)$.*
3. *If $m' \notin \{\perp, m\}$ then output (m, m') and halt.*
4. *Otherwise, start over at step 1.*

This algorithm fails to halt for $m \in U$, but the number of such m is small (= insignificant) when $|\mathcal{M}| \gg |\mathcal{H}|$.

Proof (cont.)

Proof (continued).

It may also fail even when a colliding partner m' exists if it happens that the value returned by $A(y)$ is m. (Remember, $A(y)$ is only required to return some preimage of y; we can't say which.)

However, corresponding to each such bad case is another one in which the input to the algorithm is m' instead of m. In this latter case, the algorithm succeeds, since y is the same in both cases. With this idea, we can show that the algorithm succeeds in finding a colliding partner on at least half of the messages in $\mathcal{M} - U$. □

Towards greater rigor

Katz and Lindell present rigorous definitions for what it means for collisions to be hard to find.

We'll return to those definitions later in the course.