CPSC 467b: Cryptography and Computer Security

Michael J. Fischer

Lecture 15
March 7, 2013

Digital Signatures with Special Properties (continued)

- Group signatures
- Short signatures
- Aggregate signatures

Message Digest / Cryptographic Hash Functions

Digital Signatures with Special Properties (continued)

Group signatures

A *group signature* allows a member of a group to anonymously sign for the group.

For example, a bank employee might be authorized to sign for the bank but not want his personal identity to be revealed.

A *group manager* adds group members and can determine the identify of the signer.

A *revocation manager* revokes signature authority of group members.

Properties of group signatures

Basic properties of group signatures (from Wikipedia):

Soundness and Completeness Valid signatures by group members always verify correctly, and invalid signatures always fail verification.

Unforgeable Only members of the group can create valid group signatures.

Anonymity Given a message and its signature, the identity of the individual signer cannot be determined without the group manager's secret key.

Traceability Given any valid signature, the group manager should be able to trace which user issued the signature.

Properties of group signatures (cont.)

Unlinkability Given two messages and their signatures, we cannot tell if the signatures were from the same signer or not.

No Framing Even if all other group members (and the managers) collude, they cannot forge a signature for a non-participating group member.

Unforgeable tracing verification The revocation manager cannot falsely accuse a signer of creating a signature he did not create.

Short signatures

The signature schemes we have studied all produce signatures whose length is comparable to the parameter lengths of the underlying cryptosystem, typically 1024 or 2048 or longer.

For many applications, it is desirable to have "short" signatures of lengths, say, 160 or 128 bits.

Applications of short signatures

Some applications[1]

- Electronic airline tickets.
- Serial numbers for software.
- Authorization codes.
- Electronic postage stamps.
- Electronic banking, bank notes, printed documents and certificates.
- Signing data in smart cards and other devices with limited storage capacity.

[1] From ECRYPT report D.AZTEC.7, "New Technical Trends in Asymmetric Cryptography".

Aggregate signatures

From the abstract:[2]

> *"An aggregate signature scheme is a digital signature that supports aggregation: Given n signatures on n distinct messages from n distinct users, it is possible to aggregate all these signatures into a single short signature. This single signature (and the n original messages) will convince the verifier that the n users did indeed sign the n original messages (i.e., user i signed message M_i for $i = 1, \ldots, n$)."*

[2] "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps" by D. Boneh, C. Gentry, H. Shacham, and B. Lynn, Eurocrypt 2003, LNCS, 2656, 416–432.

Message Digest / Cryptographic Hash Functions

Random functions

A *random function* from domain $\mathcal{M}$ to range $\mathcal{H}$ is a uniformly distributed element h over the space of all functions $\mathcal{M} \rightarrow \mathcal{H}$.

Intuitively, for each $m \in \mathcal{M}$, $h(m)$ is a uniformly distributed random number over $\mathcal{H}$, but for any particular h, $h(m)$ is a fixed value. If $h(m)$ is evaluated several times, the answer is the same each time.

Cryptographic use of random functions

A random function h gives a way to protect the integrity of messages.

Suppose Bob knows $h(m)$ for Alice's message m, and Bob receives m' from Alice. If $h(m') = h(m)$, then with very high probability, $m' = m$, and Bob can be assured of the integrity of m'.

The problem with this approach is that we have no succinct way of describing random functions, so there is no way for Bob to compute $h(m')$.

Message digest functions

A *message digest* (also called a *cryptographic hash* or *fingerprint*) function is a fixed (non-random) function that is designed to "look like" a random function.

The goal is to preserve the integrity-checking property of random functions: If Bob knows $h(m)$ and he receives m', then if $h(m') = h(m)$, he can reasonably assume that $m' = m$.

We now try to formalize what we require of a message digest function in order to have this property.

We also show that message digest functions do not necessarily "look random", so one should not assume such functions share other properties with random functions.

Formal definition of message digest functions

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: Given $y \in \mathcal{H}$, it is hard to find $m \in \mathcal{M}$ such that $h(m) = y$.
- Weakly collision-free: Given $m \in \mathcal{M}$, it is hard to find $m' \in \mathcal{M}$ such that $m' \neq m$ and $h(m') = h(m)$.
- Strongly collision-free: It is hard to find colliding pairs (m, m').

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?

Recall from lecture 10, 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 strong $\Rightarrow$ 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.

A randomized algorithm to enumerate 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 $\Rightarrow$ 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 that weak $\Rightarrow$ one-way (cont.)

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 that weak ⇒ one-way (cont.)

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 $\mathcal{M} - U$.