Cryptographic Hash Functions

Debdeep Mukhopadhyay

Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Kharagpur
INDIA -721302

Objectives

- **Applications**
- **Security Requirements**
  - **Randomized Algorithms**
- **Relative order of hardness**

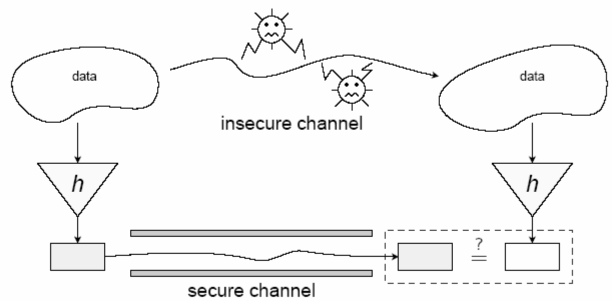
Data Integrity

- **Cryptographic Hash Function: Provides assurance of data integrity**
- **Let h be a hash function and x some data.**
- **The hash creates a *fingerprint* of the data, often referred to as the message digest.**
- **Typically, x is a large binary string**
- **The digest is a fairly short binary string, say 160 bits.**

Applications

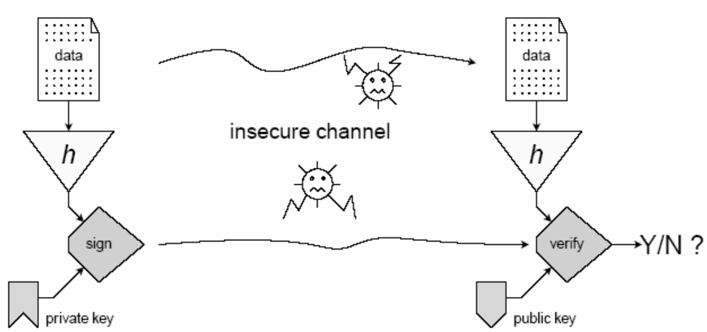
- **Say y=h(x), and y is stored in some secured place.**
- **If x is altered to say x' and if we assume that h(x)≠h(x'), then the alteration of the message is readily caught, by verifying y≠y', where y'=h(x')**
- **Used in digital signature schemes**
- **Used for message authentication codes (MAC)**

Application: Data Integrity



Comparing the digest of data sent over insecure communication channel with securely obtained original digest allows to verify integrity of the data.

Application: Digital Signatures



A Keyed Hash Function

- **Suppose we also have a key in the computation of the hash functions.**
- **$y=h_K(x)$, and the key is kept secret.**
  - **Alice and Bob share K**
  - **Alice computes y for x, using K and sends to Bob.**
  - **Bob receives x' and computes the hash value.**
  - **If the hashes match, the message is unaltered.**
  - **Note that here y is not required to be kept secret. Why?**

What is a Cryptographic Hash Family?

A *hash family* is a four-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, where the following conditions are satisfied:

1. $\mathcal{X}$ is a set of possible *messages*
2. $\mathcal{Y}$ is a finite set of possible *message digests* or *authentication tags*
3. $\mathcal{K}$, the *keyspace*, is a finite set of possible *keys*
4. For each $K \in \mathcal{K}$, there is a *hash function* $h_K \in \mathcal{H}$. Each $h_K : \mathcal{X} \to \mathcal{Y}$.

- **Note: X could be finite or infinite set, but Y is always finite**
- **If |X|=N, |Y|=M, then there are M^N possible $F^{X,Y}$ (the cardinality of the set of all functions from X to Y)**
- **Any hash family, $F \subseteq F^{X,Y}$ is called an (N,M) hash family.**

Security of Hash Functions

- There are three important properties which a hash function must satisfy.
- The properties are required for the security of the applciations.
  - Preimage
  - Second Preimage
  - Collision
- We define them one by one.

Preimage

	Preimage
Instance:	A hash function $h : \mathcal{X} \rightarrow \mathcal{Y}$ and an element $y \in \mathcal{Y}$.
Find:	$x \in \mathcal{X}$ such that $h(x) = y$.

- If the Preimage can be solved then (x,y) is a valid pair.
- A hash function for which Preimage cannot be efficiently solved is said to be preimage resistant.

Second Preimage

	Second Preimage
Instance:	A hash function $h : \mathcal{X} \rightarrow \mathcal{Y}$ and an element $x \in \mathcal{X}$.
Find:	$x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x') = h(x)$.

- **If this problem is solved, then the pair (x’,h(x)) is valid**
- **If it cannot be done efficiently then the hash is Second Preimage resistant.**

Collision

	Collision
Instance:	A hash function $h : \mathcal{X} \rightarrow \mathcal{Y}$.
Find:	$x, x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x') = h(x)$.

- **Note that if this is solved, then if (x,y) is a valid pair so is (x’,y)**
- **If not (efficiently solvable) the hash function is called collision resitant**

The Random Oracle Model

- **Captures the concept of an ideal hash function**
- **If a hash function, *h* is ideal then the only way to compute the hash of a given value is by actually computing it: i,e even if many previous values are known.**

A Non-Ideal Hash Function

- **Consider a hash function h: $Z_n \rightarrow Z_n$ which is a linear function, say**
  - **$h(x,y)=ax + by \bmod n$, a, b ε Z_n, $n \geq 2$ is a positive integer**
  - **Suppose, $h(x_1,y_1)=ax_1+by_1$, $h(x_2,y_2)=ax_2+by_2$**

$h(rx_1+sx_2 \bmod n, ry_1+sy_2 \bmod n)=$

$=rh_1(x_1,y_1)+sh_2(x_2,y_2) \bmod n$

Thus we can compute the hash of another value apart from (x_1,y_1) and (x_2,y_2) without actually computing the hash value.

We are computing the new hash value from pre-computed values

Note that we do not require the knowledge of a and b also.

This is not what is an ideal hash function according to the RO model.

What is an Oracle?

- It is not an algorithm
- neither a formula
- imagine this to be a giant book of random numbers and each page is a value x and the number written on that page is h(x)

An Independence Theorem

Suppose that $h \in \mathcal{F}^{\mathcal{X},\mathcal{Y}}$ is chosen randomly, and let $\mathcal{X}_0 \subseteq \mathcal{X}$. Suppose that the values $h(x)$ have been determined (by querying an oracle for h) if and only if $x \in \mathcal{X}_0$. Then $\mathbf{Pr}[h(x) = y] = 1/M$ for all $x \in \mathcal{X} \backslash \mathcal{X}_0$ and all $y \in \mathcal{Y}$.

- Note that the above is a conditional probability
- It states that the knowledge of the previously computed values, does not give any advantage to the future computations of h(x)
- This assumption in the RO model will be used in the complexity proofs that follow.

Algorithms in the RO model

- **These algorithms are applicable to all hash functions, since the algorithms are not dependent on the details of the hashing method.**
- **These algorithms are randomized, in the sense that they make random choices**
- **In particular they can fail, but if they succeed they are correct: *Las Vegas Algorithms***

Algorithms in the RO model

- **Worst case success probability, ε: if for *every* problem instance, the randomized algorithm returns a correct answer with probability at least ε**
- **Average case success probability: if the probability that the algorithm returns a correct answer, averaged over all problem instances, is at least ε**
- **The average success probability is averaged over all possible random choices of $F^{X,Y}$, and all possible random choices of $x \in X$ and/or $y \in Y$, if x and/or y are specified as a part of the problem instance.**

Algorithm Find-Preimage

FIND-PREIMAGE(h, y, Q)

choose any $\mathcal{X}_0 \subseteq \mathcal{X}$, $|\mathcal{X}_0| = Q$
for each $x \in \mathcal{X}_0$
do $\begin{cases} \textbf{if } h(x) = y \\ \quad \textbf{then return } (x) \end{cases}$
return (failure)

For any $\mathcal{X}_0 \subseteq \mathcal{X}$ with $|\mathcal{X}_0| = Q$, the average-case success probability : $\epsilon = 1 - (1 - 1/M)^Q$.

Algorithm Find-Second Preimage

FIND-SECOND-PREIMAGE(h, x, Q)

$y \leftarrow h(x)$
choose $\mathcal{X}_0 \subseteq \mathcal{X} \backslash \{x\}$, $|\mathcal{X}_0| = Q - 1$
for each $x_0 \in \mathcal{X}_0$
do $\begin{cases} \textbf{if } h(x_0) = y \\ \quad \textbf{then return } (x_0) \end{cases}$
return (failure)

For any $\mathcal{X}_0 \subseteq \mathcal{X} \backslash \{x\}$ with $|\mathcal{X}_0| = Q - 1$, the success probability is $\epsilon = 1 - (1 - 1/M)^{Q-1}$.

Algorithm FindCollision

FIND-COLLISION(h, Q)

choose $\mathcal{X}_0 \subseteq \mathcal{X}$, $|\mathcal{X}_0| = Q$
for each $x \in \mathcal{X}_0$
  do $y_x \leftarrow h(x)$
if $y_x = y_{x'}$ for some $x' \neq x$
  then return (x, x')
  else return (failure)

For any $\mathcal{X}_0 \subseteq \mathcal{X}$ *with* $|\mathcal{X}_0| = Q$*, the success probability of* *is*

$$\epsilon = 1 - \left(\frac{M-1}{M}\right)\left(\frac{M-2}{M}\right)\cdots\left(\frac{M-Q+1}{M}\right).$$

Relating Q and ε

$$Q \approx \sqrt{2M \ln \frac{1}{1-\epsilon}}.$$

If we take $\epsilon = .5$, then our estimate is

$$Q \approx 1.17\sqrt{M}.$$

- **So, if we hash little over sqrt(M) values, we have a 50% chance of collision**
- **Thus our algorithm is (1/2, O(sqrt(M)) algorithm**

Comparison of Security Criteria

- **Solving Collision is easier than solving Preimage or 2nd Preimage**
- **Can we reduce one problem to the other?**
- **We shall study two reductions:**
  - **Collision to 2nd Preimage**
  - **Collision to Preimage**

Proof Method

- **Reducing Collision to Preimage:**
  - **Assume that Preimage can be solved using a randomized algorithm**
  - **Show that then the Collision can be solved.**
- **$\text{Collision}_{\text{Hardness}} << \text{Preimage}_{\text{Hardness}}$**
- **Resistance against Collision => Preimage Resistance**

The first reduction

Collision-To-Second-Preimage(h)

external Oracle-2nd-Preimage
choose $x \in \mathcal{X}$ uniformly at random
if Oracle-2nd-Preimage$(h, x) = x'$
  then return (x, x')
  else return (failure)

- **Oracle-2nd-Preimage is an (ε,q) algorithm.**
- **Since it is a Las-Vegas algorithm, if it gives an answer it will be correct. Thus, x≠x' and h(x)=h(x'). Thus the collision is also found.**
- **Thus Collision-to-second-preimage is also an (ε,q) Las-Vegas algorithm**

The second reduction

Collision-To-Preimage(h)

external Oracle-Preimage
choose $x \in \mathcal{X}$ uniformly at random
$y \leftarrow h(x)$
if (Oracle-Preimage$(h, y) = x'$) **and** $(x' \neq x)$
  then return (x, x')
  else return (failure)

- **Assume that Oracle-Preimage is a (1,Q) Las Vegas algorithm**
- **We will make some weak assumptions on the size of X and Y, |X|≥2|Y|**

Reduction

Suppose $h : \mathcal{X} \rightarrow \mathcal{Y}$ is a hash function where $|\mathcal{X}|$ and $|\mathcal{Y}|$ are finite and $|\mathcal{X}| \geq 2|\mathcal{Y}|$. Suppose ORACLE-PREIMAGE *is a $(1, Q)$-algorithm for* **Preimage**, *for the fixed hash function h. Then* COLLISION-TO-PREIMAGE *is a $(1/2, Q+1)$-algorithm for* **Collision**, *for the fixed hash function h.*

- **Proof discussed in class.**

Point to Ponder

- **If the OraclePreimage has a success probability of ε<1, what is the minimum probability of success of CollisionToPreimage Algorithm?**

Further Reading

- **Douglas Stinson, *Cryptography Theory and Practice, 2nd Edition*, Chapman & Hall/CRC**
- **Phillip Rogaway and Thomas Shrimpton, Cryptographic Hash-Function Basics: Definitions, Implications, and Separations for Preimage Resistance, Second-Preimage Resistance, and Collision Resistance, Fast Software Encryption 2004.**

Next Days Topic

- **Cryptographic Hash Functions (contd.)**