Cryptographic applications of number-theoretic assumptions
One-way functions

Foundations of Cryptography
Computer Science Department
Wellesley College

Fall 2016

Table of contents

Introduction

One-Way Functions

Hash functions
One-way functions and permutations

- **One-way function** are the minimal cryptographic primitive.
- They are both necessary and sufficient for all private-key construction seen to date.
- Informally, a function is **one-way** if it is easy to compute but hard to invert.

Formally

The inverting experiment $\text{Invert}_{A,f}(n)$:

1. Choose input $x \leftarrow \{0,1\}^n$. Compute $y := f(x)$.
2. $A$ is given $1^n$ and $y$ as input, and outputs $x'$.
3. The output of the experiment is defined to be 1 if and only if $f(x') = y$.

**Definition 8.72.** A function $f : \{0,1\}^* \rightarrow \{0,1\}^*$ is **one-way** if the following two conditions hold:

1. (Easy to compute:) There exists a polynomial-time algorithm that on input $x$ outputs $f(x)$.
2. (Hard to invert:) For all probabilistic polynomial-time algorithms $A$ there exists a negligible function $\text{negl}$ such that

   $$\Pr[\text{Invert}_{A,f}(n) = 1] \leq \text{negl}(n).$$
Framework for a hard problem

Let Gen be a polynomial-time algorithm that, on input $1^n$, outputs $(N, p, q)$ where $N = pq$, and $p$ and $q$ are $n$-bit primes except with probability negligible in $n$.

The factoring experiment Factor$_{A,Gen}(n)$:

1. Run Gen$(1^n)$ to obtain $(N, p, q)$.
2. $A$ is given $N$, and outputs $p', q' > 1$.
3. The output of the experiment is defined to be 1 if $p' \cdot q' = 1$, and 0 otherwise.

Definition 8.45. We say that factoring problem is hard relative to Gen if for all probabilistic polynomial-time algorithms $A$ there exists a negligible function negl such that

$$\Pr[\text{Factor}_{A,Gen}(n) = 1] \leq \text{negl}(n).$$

One-way functions and permutations

- The factoring assumption is simply the assumption that there exists a Gen relative to which factoring is hard.
- So is factoring a candidate for a one-way function?
Constructing a candidate “one-way” function $f_{Gen}$

*Algorithm 8.73.*
*Algorithm for computing $f_{Gen}$*

**Input:** String $x$
**Output:** String $N$

```
compute \((N, p, q) := \text{Gen}(1^n; x)\)
// i.e., run Gen\((1^n)\) using $x$ as the random tape
return $N$
```

*Since Gen runs in polynomial-time, there exists a polynomial $p$ such that the number of random bits the algorithm uses on input $1^n$ is at most $p(n)$. For simplicity, we assumption Gen uses exactly $p(n)$ random bits and is strictly increasing.*

**One-way**

*Theorem 8.74* If the factoring problem is hard relative to Gen, then $f_{Gen}$ is a one-way function.

**Proof.** Certainly $f_{Gen}$ is polynomial-time to compute. We show that $f_{Gen}$ is hard to invert.

Let $\mathcal{A}$ be a PPT adversary. We show

$$\Pr[\text{Invert}_{\mathcal{A}, f}(n) = 1] \leq \text{negl}(n)$$

for some negligible function negl.
One-way

Consider the following adversary $\mathcal{A}'$ against the factoring problem:

**Adversary $\mathcal{A}'$**

1. On input $N$, set $n' = p(n)$ and run $\mathcal{A}$ on inputs $1^{n'}$ and $N$.
2. When $\mathcal{A}$ returns $x$, run $\text{Gen}(1^{n'}; x)$ to obtain $(N, p, q)$.
3. Return $p, q$.

Note that the view of $\mathcal{A}$ when run as a subroutine of $\mathcal{A}'$ is identical to the view of $\mathcal{A}$ in the experiment $\text{Invert}_{\mathcal{A}, f}(n)$. Furthermore $\mathcal{A}'$ factors $N$ precisely when $\mathcal{A}$ successfully inverts $f_{\text{Gen}}$. Thus,

$$\Pr[\text{Invert}_{\mathcal{A}, f}(n) = 1] = \Pr[\text{Factor}_{\mathcal{A}', \text{Gen}}(n) = 1] \leq \text{negl}(n)$$

for some negligible function $\text{negl}$ since the factoring problem is hard relative to $\text{Gen}$.

---

Hash functions revisited

- Recall that *hash functions* take arbitrary-length strings and *compress* them into shorter strings.
- Also recall *hash collisions are bad*.
- Previously we presented heuristic constructions of collision-resistant hash-functions, but gave no proofs that the resulting hash functions were secure under more basic assumptions.*

*All that is about to change.
First a reminder: Collision experiments & resistance

The collision-finding experiment $\text{Hash-coll}_{A, \Pi}(n)$:

1. A key $s$ is generated by running $\text{Gen}(1^n)$.
2. The adversary $A$ is given $s$ and outputs $x, x'$. (If $\Pi$ is a fixed length hash function for inputs of length $\ell'(n)$ then we require $x, x' \in \{0, 1\}^{\ell'(n)}$.)
3. The output of the experiment is defined to be 1 if and only if $x \neq x'$ and $H^s(x) = H^s(x')$. In such a case we say that $A$ has found a collision.

Definition 5.2. A hash function $\Pi = (\text{Gen}, H)$ is collision resistant if for all probabilistic polynomial-time adversaries $A$ there exists a negligible function $\text{negl}$ such that

$$\Pr[\text{Hash-coll}_{A, \Pi}(n) = 1] \leq \text{negl}(n).$$

Constructing collision-resistant hash functions

Construction 8.78
Let $\mathcal{G}$ be a polynomial-time algorithm that on input $1^n$ outputs a cyclic group $\mathbb{G}$ of prime order $q$ (with $n = \|q\|$) and generator $g$. Define a fixed-length hash function $(\text{Gen}, H)$ as follows:

- **Gen**: On input $1^n$, run $\mathcal{G}(1^n)$ to obtain $(\mathbb{G}, q, g)$ and then select $h \leftarrow \mathbb{G}$. Output $s := (\mathbb{G}, q, g, h)$.
- **H**: given a key $s = (\mathbb{G}, q, g, h)$ and input $(x_1, x_2) \in \mathbb{Z}_q \times \mathbb{Z}_q$, output $H^s(x_1, x_2) := g^{x_1} h^{x_2}$. 
Collision-resistant

**Theorem 8.79.** If the discrete logarithm problem is hard relative to \( \mathcal{G} \), the Construction 8.78 is a fixed-length collision-resistant hash function.

**Proof.** Let \( \Pi = (\text{Gen}, H) \) as in Construction 8.78, and let \( \mathcal{A} \) be a PPT algorithm with

\[ \epsilon \overset{\text{def}}{=} \Pr[\text{Hash-coll}_{\mathcal{A}, \Pi}(n) = 1] \]

We show how \( \mathcal{A} \) can be used by an algorithm \( \mathcal{A}' \) to solve the discrete logarithm problem with success probability \( \epsilon \).

---

**Recall the discrete logarithm problem:**

The discrete logarithm experiment \( \text{Dlog}_{\mathcal{A}, \mathcal{G}}(n) \):

1. Run \( \mathcal{G}(1^n) \) to obtain \((\mathcal{G}, q, g)\), where \( \mathcal{G} \) is a cyclic group of order \( q \) (with \( \lVert q \rVert = n \)), and \( g \) is a generator of \( \mathcal{G} \).
2. Choose \( h \leftarrow \mathcal{G} \). (This can be done by choosing \( x' \leftarrow \mathbb{Z}_q \) and set \( h := g^{x'} \)).
3. \( \mathcal{A} \) is given \( \mathcal{G}, q, g, h \), and outputs \( x \in \mathbb{Z}_q \).
4. The output of the experiment is defined to be 1, if \( g^x = h \), and 0 otherwise.

**Definition 7.59** We say that the discrete logarithm problem is hard relative to \( \mathcal{G} \) if for all probabilistic polynomial-time algorithms \( \mathcal{A} \) there exists a negligible function \( \text{negl} \) such that

\[ \Pr[\text{Dlog}_{\mathcal{A}, \mathcal{G}}(n) = 1] \leq \text{negl}(n). \]
Algorithm $A'$:
The algorithm is given $G, q, g, h$ as input.

1. Let $s := \langle G, q, g, h \rangle$. Run $A(s)$ and obtain output $x$ and $x'$.
2. If $x \neq x'$ and $H^s(x) = H^s(x')$ then:
   2.1 If $h = 1$ return 0.
   2.2 Otherwise, parse $x$ as $(x_1, x_2)$ and parse $x'$ as $(x'_1, x'_2)$. Return $(x_1 - x'_1), (x_2 - x'_2)^{-1} \mod q$.

Clearly, $A'$ runs in polynomial time. Furthermore, the input $s$ given to $A$ when run as a subroutine by $A'$ is distributed exactly as in experiment Hash-coll$_{A, \Pi}$. So with probability precisely $\epsilon(n)$ there is a collision.

We claim that whenever there is a collision, $A'$ returns the correct answer $\log_g h$.

If $h = 1$, then $\log_g h = 0$ which is previously what $A'$ returns.

Otherwise, the collision implies

$$H^s(x_1, x_2) = H^s(x'_1, x'_2) \Rightarrow g^{x_1} h^{x_2} = g^{x'_1} h^{x'_2} \Rightarrow g^{x_1 - x'_1} = h^{x'_2 - x_2}.$$ 

If $x'_2 - x_2 = 0 \mod q$, then $g^{x_1 - x'_1} = h^{x'_2 - x_2} = h^0 = 1$ and $x_1 - x'_1 = 0 \mod q$. But then $x = (x_1, x_2) = (x'_1, x'_2) = x'$ in contradiction. Thus, $x'_2 - x_2 \neq 0 \mod q$ and has an inverse.

$$g^{(x_1 - x'_1) \cdot [(x'_2 - x_2)^{-1} \mod q]} = (h^{(x'_2 - x_2)})^{[(x'_2 - x_2)^{-1} \mod q]} = h^1 = h,$$

and so

$$\log_g h = [(x_1 - x'_1), (x_2 - x'_2)^{-1} \mod q]$$
**Theorem 8.80.** If there exists a probability polynomial-time algorithm $G$ relative to which the discrete logarithm problem is hard,

then there exists a collision resistant hash function.