Cryptography

Corso di Laurea Magistrale in Informatica

Algebra, Number Theory and Related Assumptions

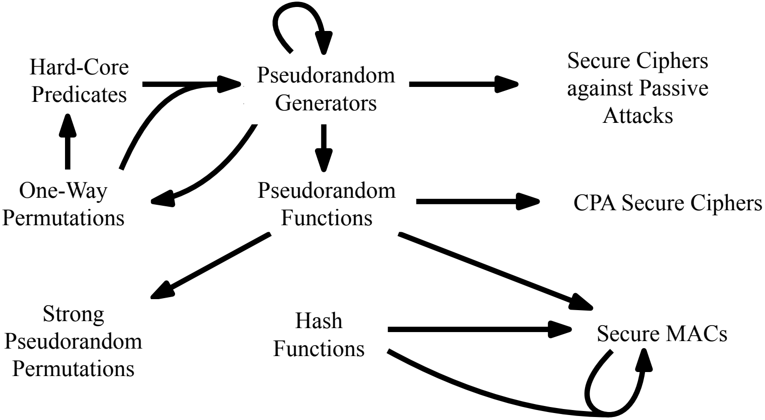
Ugo Dal Lago

ALMA MATER STUDIORUM UNIVERSITÀ DI BOLOGNA

Inria informatiques mathématiques

Academic Year 2021-2022

Where Were We?



Constructing One-Way Functions

- We have given only a few examples of "presumed" one-way functions, without further discussion of their nature.

Constructing One-Way Functions

- We have given only a few examples of "presumed" one-way functions, without further discussion of their nature.
- The most interesting one-way functions are undoubtedly those of **"mathematical" kind**.
  - In particular, those coming from number theory and algebra.
- It's time to see what it is all about.
  - We will quickly recall the necessary notions of algebra and number theory, as we need them.
  - A cryptography course designed for maths students, by the way, would start right here.

Constructing One-Way Functions

- We have given only a few examples of "presumed" one-way functions, without further discussion of their nature.
- The most interesting one-way functions are undoubtedly those of **"mathematical" kind**.
  - In particular, those coming from number theory and algebra.
- It's time to see what it is all about.
  - We will quickly recall the necessary notions of algebra and number theory, as we need them.
  - A cryptography course designed for maths students, by the way, would start right here.
- The assumptions we will see will find *direct* application in public key cryptography, which we will discuss later.

Divisors, Primes, etc.

- $\mathbb{Z}$ is the set of the integers, while $\mathbb{N}$ is the set of natural numbers.
- Given two elements $a, b \in \mathbb{Z}$, we write $a|b$ when there exists $c \in \mathbb{Z}$ such that $ac = b$.
- When $a|b$ and a is positive, we say that a is *divisor* of b, and, when $a \notin \{1, b\}$, a is said to be *factor* of b.
- A number $p \in \mathbb{N}$ with $p > 1$ is called *prime* if it has no factors, otherwise is called *composite*.
- Given $n, m \in \mathbb{N}$ with $m > 1$, let us denote by $n \mod m$ the remainder of n upon division by m.

Lemma

If $n, m \in \mathbb{N}$ and $m > 1$, then n is invertible modulo m (there exists p such that $np \mod m = 1$) whenever $gcd(n, m) = 1$, i. e. when n, m are coprime.

Primes and Factoring

- Given a natural number $N \in \mathbb{N}$, we ask ourselves whether it is difficult to determine two integers $p, q \in \mathbb{N}$ such that $N = pq$.
  - There are algorithms that take time $O(\sqrt{N}(\lg(N))^k)$, i.e. exponential over the length of N.
  - This is a problem that has nothing to do with cryptography, but which mathematicians have been studying for centuries.

Primes and Factoring

- Given a natural number $N \in \mathbb{N}$, we ask ourselves whether it is difficult to determine two integers $p, q \in \mathbb{N}$ such that $N = pq$.
  - There are algorithms that take time $O(\sqrt{N}(\lg(N))^k)$, i.e. exponential over the length of N.
  - This is a problem that has nothing to do with cryptography, but which mathematicians have been studying for centuries.
- We can therefore define the following experiment:

  $\mathsf{wFactor}_A(n)$:
  $(x, y) \leftarrow \mathbb{N} \times \mathbb{N}$ with $|x| = |y| = n$;
  $N \leftarrow x \cdot y$;
  $(z, w) \leftarrow A(N)$;
  Result: $z \cdot w = N$

- As usual, our assumption could be that for every PPT A there exists $\varepsilon \in \mathcal{NGL}$ such that

$$Pr(\mathsf{wFactor}_A(n) = 1) = \varepsilon(n)$$

Primes and Factoring

- However, the assumption we have just made *does not hold.*
  - With probability equal to $\frac{3}{4}$ the number N will be even, because it is enough that one between x and y is even for N to be even.
  - Factoring an even number is very simple.

Primes and Factoring

- However, the assumption we have just made *does not hold.*
  - With probability equal to $\frac{3}{4}$ the number N will be even, because it is enough that one between x and y is even for N to be even.
  - Factoring an even number is very simple.
- It must be guaranteed that N is not (on average) trivially factorable.
  - An interesting idea is to modify wFactor so that x and y are always and only prime numbers (representable in n bits).
  - How do we generate prime numbers randomly?
  - What is the probability that a certain prime number p will be generated?

Generating Primes Efficiently

- A possible way to generate prime numbers, that can be represented in exactly n bits, is to proceed by trial and error:

  for $i \leftarrow 1$ to t **do**
      $r \leftarrow \{0,1\}^{n-1}$;
      $p \leftarrow 1||r$;
      if p *is prime* **then**
          Result: p

  Result: `fail`

Generating Primes Efficiently

- A possible way to generate prime numbers, that can be represented in exactly n bits, is to proceed by trial and error:

  for $i \leftarrow 1$ to t **do**
      $r \leftarrow \{0,1\}^{n-1}$;
      $p \leftarrow 1||r$;
      if p *is prime* **then**
          Result: p
  Result: `fail`

- However, there are two questions that need to be answered.

1. **How to value the parameter t?**
   - We would like to define t as a polynomial in n, for efficiency reasons.
   - We would like that for such a value of t the probability of obtaining `fail` is negligible (in n).
2. **How to test the primality of a number?**
   - We would like to be able to test whether p is prime in polynomial time.

How to Value t?

Theorem

There exists a constant c such that for every $n > 1$ the number of primes that can be represented in exactly n bits is at least equal to $\frac{c \cdot 2^{n-1}}{n}$.

- At each iteration, therefore, the probability that p is actually prime will be at least equal to:

$$\frac{c \cdot 2^{n-1}/n}{2^{n-1}} = \frac{c \cdot 2^{n-1}}{n \cdot 2^{n-1}} = \frac{c}{n}$$

- Therefore, if $t = \frac{n^2}{c}$, the probability of getting `fail` will be at most equal to

$$\left(1 - \frac{c}{n}\right)^t = \left(\left(1 - \frac{c}{n}\right)^{\frac{n}{c}}\right)^n \leq (e^{-1})^n = e^{-n}$$

whenever $n \geq c$

How to Test the Primality of a Number in Polynomial Time?

- There are **deterministic** algorithms for primality testing that take polynomial time.
  - The first one, the so-called *AKS algorithm*, has been developed by two Master's students of IIT-Kanpur, India.
  - The degree of the polynomial is *too high* to consider these algorithms interesting from a practical point of view.
- Instead, the so-called **Miller-Rabin test** is probabilistic, but it is PPT.
  - If the input is a prime number, the Miller-Rabin test returns `prime` with probability 1.
  - If the input p is a composite number, the Miller-Rabin test returns `composite` with probability $1 - \varepsilon(|p|)$, where $\varepsilon \in \mathcal{NGL}$.
  - The degree of the polynomial that upper bounds the complexity of the test is relatively low.

The Assumption, Properly Formalised

- The assumption we are trying to define will be parameterised on an algorithm, called $\mathsf{GenModulus}$ which, on input 1^n, outputs a triple (N, p, q) where $N = pq$ and p, q are primes with n bits.
- The experiment $\mathsf{wFactor}$ becomes the following one:

  $\mathsf{Factor}_{A,\mathsf{GenModulus}}(n)$:
  $(N, p, q) \leftarrow \mathsf{GenModulus}(1^n)$;
  $(r, s) \leftarrow A(N)$;
  Result: $r \cdot s = N$

- We say that *factoring is hard relative to* $\mathsf{GenModulus}$ iff for every algorithm A which is PPT there exists a negligible function $\varepsilon \in \mathcal{NGL}$ such that

$$Pr(\mathsf{Factor}_{A,\mathsf{GenModulus}}(n) = 1) = \varepsilon(n)$$

- This assumption is sufficient to obtain a one-way function, but not to prove the security of public-key schemes.

Group Theory

- A **group** is an algebraic structure $(\mathbb{G}, \circ)$ where $\circ$ is a binary operation that is associative, with identity e and where every $g \in \mathbb{G}$ has an inverse g^{-1}.
- A finite group $(\mathbb{G}, \circ)$ is said to have **order** equal to $|\mathbb{G}|$.
- A group is said to be **abelian** if $\circ$ is a commutative operation.
- The binary operation is often denoted:
  - With the addition symbol $+$, in this case the group is called **additive** and if $m \in \mathbb{N}$ we can use the notation

    $$mg = \underbrace{g + \cdots + g}_{m \text{ times}}$$

  - With the multiplication symbol $\cdot$, in this case the group is called **multiplicative**, and if $m \in \mathbb{N}$ we can write

    $$g^m = \underbrace{g \cdot \ldots \cdot g}_{m \text{ times}}$$

Exponentiation

- The computation of mg or g^m can be performed in a number of operations which is polynomial in $|m|$, i.e. logarithmic in m.
  - Just proceed considering the binary representation of m. For example:

    $$g^{11} = g^8 \cdot g^2 \cdot g^1 = g^{2^3} \cdot g^{2^1} \cdot g^{2^0}$$

    and each of the factors, which are at most $|m|$, can be computed in time that is linear in $|m|$.

Exponentiation

- The computation of mg or g^m can be performed in a number of operations which is polynomial in $|m|$, i.e. logarithmic in m.
  - Just proceed considering the binary representation of m. For example:

$$g^{11} = g^8 \cdot g^2 \cdot g^1 = g^{2^3} \cdot g^{2^1} \cdot g^{2^0}$$

    and each of the factors, which are at most $|m|$, can be computed in time that is linear in $|m|$.

Theorem

If $(\mathbb{G}, \cdot)$ has order m, then for each $g \in \mathbb{G}$, it is true that $g^m = 1_{\mathbb{G}}$.

Corollary

If $(\mathbb{G}, \cdot)$ has order $m > 1$, then for every $g \in \mathbb{G}$ and for every i, $g^i = g^{[i \mod m]}$.

Finite Groups Examples

- The set $\mathbb{Z}_N = \{0, \ldots, N-1\}$ is a group if the underlying operation is addition modulo N, i.e. the map matching (a, b) with $a + b \mod N$.
  - The identity is 0;
  - The inverse of $n \in \mathbb{Z}_N$ is $N - n$.

Finite Groups Examples

- The set $\mathbb{Z}_N = \{0, \ldots, N-1\}$ is a group if the underlying operation is addition modulo N, i.e. the map matching (a, b) with $a + b \mod N$.
  - The identity is 0;
  - The inverse of $n \in \mathbb{Z}_N$ is $N - n$.
- The set $\mathbb{Z}_N$ becomes a group with multiplication modulo N when:
  - We eliminate 0 (which is not invertible) from the group.
  - N is prime. This guarantees that every $1 < n < N$ is invertible modulo N.

Finite Groups Examples

- The set $\mathbb{Z}_N = \{0, \ldots, N-1\}$ is a group if the underlying operation is addition modulo N, i.e. the map matching (a, b) with $a + b \mod N$.
  - The identity is 0;
  - The inverse of $n \in \mathbb{Z}_N$ is $N - n$.
- The set $\mathbb{Z}_N$ becomes a group with multiplication modulo N when:
  - We eliminate 0 (which is not invertible) from the group.
  - N is prime. This guarantees that every $1 < n < N$ is invertible modulo N.
- If we consider $\mathbb{Z}_N$ with composite N, is there a way to make this set a group with respect to the multiplication modulo N:
  - Just consider $\mathbb{Z}_N^* \subseteq \mathbb{Z}_N$ defined as $\{n \in \mathbb{Z}_N \mid gcd(n, N) = 1\}$.

Finite Groups Examples

- The set $\mathbb{Z}_N = \{0, \ldots, N-1\}$ is a group if the underlying operation is addition modulo N, i.e. the map matching (a, b) with $a + b \mod N$.
  - The identity is 0;
  - The inverse of $n \in \mathbb{Z}_N$ is $N - n$.
- The set $\mathbb{Z}_N$ becomes a group with multiplication modulo N when:
  - We eliminate 0 (which is not invertible) from the group.
  - N is prime. This guarantees that every $1 < n < N$ is invertible modulo N.
- If we consider $\mathbb{Z}_N$ with composite N, is there a way to make this set a group with respect to the multiplication modulo N:
  - Just consider $\mathbb{Z}_N^* \subseteq \mathbb{Z}_N$ defined as $\{n \in \mathbb{Z}_N \mid gcd(n, N) = 1\}$.
- All groups considered are *abelian* groups.

On the Cardinality of $\mathbb{Z}_N^*$

- The function that associates to every natural $N > 1$ the cardinality of $\mathbb{Z}_N^*$ is called *Euler function* and is denoted by Φ:

$$\Phi(N) = |\mathbb{Z}_N^*|$$

- Of course, $1 \leq \Phi(N) < N$. But can we say something more?
  - If N is a prime number p, then $\Phi(N) = p - 1$ because every n between 1 and $p - 1$ is coprime with p, i.e. $gcd(n, p) = 1$.
  - If N is the product of two primes p and q, then $gcd(a, N) \neq 1$ precisely when $p|a$ or $q|a$. Therefore:

$$\begin{aligned}\Phi(N) &= N - 1 - (p - 1) - (q - 1) = pq - p - q + 1 \\ &= p(q - 1) - 1(q - 1) = (p - 1)(q - 1)\end{aligned}$$

Residue Classes and Exponentiation

Theorem

Let $N > 1$. For every natural $e > 0$, we define $f_e : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ assuming $f_e(x) = x^e \mod N$. If $gcd(e, \Phi(N)) = 1$, then f_e is a permutation. Moreover, if d is the inverse of e (modulo $\Phi(N)$), then f_d is the inverse of f_e.

Residue Classes and Exponentiation

Theorem

Let $N > 1$. For every natural $e > 0$, we define $f_e : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ assuming $f_e(x) = x^e \mod N$. If $gcd(e, \Phi(N)) = 1$, then f_e is a permutation. Moreover, if d is the inverse of e (modulo $\Phi(N)$), then f_d is the inverse of f_e.

- Let us observe that, given e and N, the value of $f_e(x)$ is efficiently computable from x.
  - Just apply the exponentiation algorithm.

Residue Classes and Exponentiation

Theorem

Let $N > 1$. For every natural $e > 0$, we define $f_e : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ assuming $f_e(x) = x^e \mod N$. If $gcd(e, \Phi(N)) = 1$, then f_e is a permutation. Moreover, if d is the inverse of e (modulo $\Phi(N)$), then f_d is the inverse of f_e.

- Let us observe that, given e and N, the value of $f_e(x)$ is efficiently computable from x.
  - Just apply the exponentiation algorithm.
- We also observe that, given e and $\Phi(N)$, the inverse d of e modulo $\Phi(N)$ is efficiently computable.
  - The inversion modulo any integer is a problem that can be handled.

Residue Classes and Exponentiation

Theorem

Let $N > 1$. For every natural $e > 0$, we define $f_e : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ assuming $f_e(x) = x^e \mod N$. If $gcd(e, \Phi(N)) = 1$, then f_e is a permutation. Moreover, if d is the inverse of e (modulo $\Phi(N)$), then f_d is the inverse of f_e.

- Let us observe that, given e and N, the value of $f_e(x)$ is efficiently computable from x.
  - Just apply the exponentiation algorithm.
- We also observe that, given e and $\Phi(N)$, the inverse d of e modulo $\Phi(N)$ is efficiently computable.
  - The inversion modulo any integer is a problem that can be handled.
- Finally, we observe that, given N as the product of two primes p and q, the value of $\Phi(N)$ is *not easily* computable.
  - It would be if we could (efficiently) factorise N.

RSA Assumption

- In the so-called RSA Assumption we state that *it is hard to invert* $f_e : \mathbb{Z}_N^* \to \mathbb{Z}_N^*$ when N is the product of two primes p and q, and $gcd(e, \Phi(N)) = 1$.
- Unlike the assumption on factoring, however, the problem we are assuming to be hard *becomes easy* if
  - we know not only N but also p and q;
  - or if we know an inverse d of e (modulo $\Phi(N)$).
- The RSA assumption is parametrized on a routine GenRSA which, given an input 1^n outputs:
  - A natural N which is the product of two primes p and q with $|p| = |q| = n$.
  - A natural e such that $gcd(e, \Phi(N)) = 1$.
  - A natural d such that $ed \mod \Phi(N) = 1$.

RSA Assumption

- The experiment RSAInv is parameterized on an adversary and on the routine GenRSA:

$$
\begin{array}{l}
\mathsf{RSAInv}_{A,\mathsf{GenRSA}}(n): \\
(N, e, d) \leftarrow \mathsf{GenRSA}(1^n); \\
y \leftarrow \mathbb{Z}_N^*; \\
x \leftarrow A(N, e, y); \\
\textbf{Result: } x^e \mod N = y
\end{array}
$$

RSA Assumption

- The experiment RSAInv is parameterized on an adversary and on the routine GenRSA:

  $\mathsf{RSAInv}_{A,\mathsf{GenRSA}}(n)$:
  $(N, e, d) \leftarrow \mathsf{GenRSA}(1^n)$;
  $y \leftarrow \mathbb{Z}_N^*$;
  $x \leftarrow A(N, e, y)$;
  Result: $x^e \mod N = y$

- We say that *the RSA problem is hard relative to* GenRSA iff for every adversary A that is PPT there exists $\varepsilon \in \mathcal{NGL}$ such that

$$Pr(\mathsf{RSAInv}_{A,\mathsf{GenRSA}}(n) = 1) = \varepsilon(n)$$

RSA Assumption

- The experiment RSAInv is parameterized on an adversary and on the routine GenRSA:

  $\mathsf{RSAInv}_{A,\mathsf{GenRSA}}(n)$:
  $(N, e, d) \leftarrow \mathsf{GenRSA}(1^n)$;
  $y \leftarrow \mathbb{Z}_N^*$;
  $x \leftarrow A(N, e, y)$;
  Result: $x^e \mod N = y$

- We say that *the RSA problem is hard relative to* GenRSA iff for every adversary A that is PPT there exists $\varepsilon \in \mathcal{NGL}$ such that

$$Pr(\mathsf{RSAInv}_{A,\mathsf{GenRSA}}(n) = 1) = \varepsilon(n)$$

- As we have already argued, if A could factorize N, it would be able to compute $\Phi(N)$ and therefore compute d, thus inverting f_e.

How to Construct GenRSA?

- How can we construct the GenRSA algorithm so that it is efficiently computable?
- Of course, GenModulus helps, but obviously is not enough.
- For example, we could proceed as follows:

  $(N, p, q) \leftarrow \mathsf{GenModulus}(1^n)$;
  $M \leftarrow (p-1)(q-1)$;
  $e \leftarrow \{1, \ldots, M\}$ such that $gcd(e, M) = 1$;
  $d \leftarrow e^{-1} \mod M$;
  Result: (N, e, d)

- If GenRSA is constructed in this way from GenModulus, it is possible to prove that from the Assumption RSA *follows* the Factoring Assumption.

RSA Assumption $\longrightarrow$ Factoring Assumption

Cyclic Groups

- We consider a finite multiplicative group $(G, \cdot)$, one of its elements $g \in \mathbb{G}$ and we construct

$$\langle g \rangle = \{g^0, g^1, g^2, \ldots\} \subseteq \mathbb{G}$$

Cyclic Groups

- We consider a finite multiplicative group $(G, \cdot)$, one of its elements $g \in \mathbb{G}$ and we construct

$$\langle g \rangle = \{g^0, g^1, g^2, \ldots\} \subseteq \mathbb{G}$$

- We know that $g^m = 1_{\mathbb{G}}$, so we can certainly write that $\langle g \rangle = \{g^1, \ldots, g^m\}$.

Cyclic Groups

- We consider a finite multiplicative group $(G, \cdot)$, one of its elements $g \in \mathbb{G}$ and we construct

$$\langle g \rangle = \{g^0, g^1, g^2, \ldots\} \subseteq \mathbb{G}$$

- We know that $g^m = 1_{\mathbb{G}}$, so we can certainly write that $\langle g \rangle = \{g^1, \ldots, g^m\}$.
- However, there might be an $i < m$ such that $g^i = 1_{\mathbb{G}}$. For obvious reasons$\langle g \rangle = \{g^1, \ldots, g^i\}$, and therefore $\langle g \rangle$ contains at most i elements.
  - Actually, it has *exactly* i elements; if $1 \leq k < j < i$, then

$$g^j = g^k \implies g^j \cdot g^{-k} = 1_{\mathbb{G}} \implies g^{j-k} = 1_{\mathbb{G}}.$$

Cyclic Groups

- We consider a finite multiplicative group $(G, \cdot)$, one of its elements $g \in \mathbb{G}$ and we construct

$$\langle g \rangle = \{g^0, g^1, g^2, \dots\} \subseteq \mathbb{G}$$

- We know that $g^m = 1_{\mathbb{G}}$, so we can certainly write that $\langle g \rangle = \{g^1, \dots, g^m\}$.
- However, there might be an $i < m$ such that $g^i = 1_{\mathbb{G}}$. For obvious reasons$\langle g \rangle = \{g^1, \dots, g^i\}$, and therefore $\langle g \rangle$ contains at most i elements.
  - Actually, it has *exactly* i elements; if $1 \leq k < j < i$, then

$$g^j = g^k \implies g^j \cdot g^{-k} = 1_{\mathbb{G}} \implies g^{j-k} = 1_{\mathbb{G}}.$$

- The **order** of $g \in \mathbb{G}$ is the smallest natural i such that $g^i = 1$, i.e. the cardinality of $\langle g \rangle$.
- A finite group $(\mathbb{G}, \cdot)$ is said to be **cyclic** if there exists $g \in \mathbb{G}$ with $\langle g \rangle = \mathbb{G}$. Such a g is said **generator** of $\mathbb{G}$.

Cyclic Groups and Order

Lemma

If $\mathbb{G}$ has order m and $g \in \mathbb{G}$ has order i, then $i|m$

Cyclic Groups and Order

Lemma

If $\mathbb{G}$ has order m and $g \in \mathbb{G}$ has order i, then $i|m$

- If this were not so, we would have that $m = ik + j$ with $j < i$. But then

$$g^j = g^{j+ik-ik} = g^{m-ik} = g^m (g^i)^{-k} = 1_{\mathbb{G}} (1_{\mathbb{G}})^{-k} = 1_{\mathbb{G}}$$

Cyclic Groups and Order

Lemma

If $\mathbb{G}$ has order m and $g \in \mathbb{G}$ has order i, then $i|m$

- If this were not so, we would have that $m = ik + j$ with $j < i$. But then

$$g^j = g^{j+ik-ik} = g^{m-ik} = g^m (g^i)^{-k} = 1_{\mathbb{G}} (1_{\mathbb{G}})^{-k} = 1_{\mathbb{G}}$$

Theorem

If $\mathbb{G}$ has prime order then $\mathbb{G}$ is cyclic and every $g \in \mathbb{G}$ with $g \neq 1_{\mathbb{G}}$ generates $\mathbb{G}$.

Discrete Logarithm Assumption

- If $\mathbb{G}$ is a cyclic multiplicative group, then there exists a "natural" biunivocal correspondence between $\mathbb{G}$ and $\mathbb{Z}_{|\mathbb{G}|}$.
  - Every $h \in \mathbb{G}$ can be matched with a *unique* $x \in \mathbb{Z}_{|\mathbb{G}|}$ that is the one such that $g^x = h$. Let us call such x the **discrete logarithm** of h with respect to g, which we write $\log_g h$.
- The *discrete logarithm problem* is simply the problem of computing $log_g h$ given a cyclic group $\mathbb{G}$, a generator g for $\mathbb{G}$ and a random element h.
- The experiment by which we will formalise the assumption of the discrete logarithm is parametrized, as usual, by a routine GenCG which, given 1^n, constructs a group $\mathbb{G}$, of order q, with $|q| = n$, and a generator $g \in \mathbb{G}$.

Discrete Logarithm Assumption

- The experiment DLog is defined as follows:

  $\mathsf{DLog}_{A,\mathsf{GenCG}}(n)$:
  $(\mathbb{G}, q, g) \leftarrow \mathsf{GenCG}(1^n)$;
  $h \leftarrow \mathbb{G}$;
  $x \leftarrow A(\mathbb{G}, q, g, h)$;
  Result: $g^x = h$

- As usual, let us say that the *discrete logarithm assumption is valid with respect to* GenCG iff for every PPT adversary A there exists $\varepsilon \in \mathcal{NGL}$ such that that

$$Pr(\mathsf{DLog}_{A,\mathsf{GenCG}}(n) = 1) = \varepsilon(n)$$

Computational Diffie-Helmann Assumption

- Given a cyclic group $\mathbb{G}$ and a generator $g \in \mathbb{G}$ for it, let us define the function $DH_g : \mathbb{G} \times \mathbb{G} \to \mathbb{G}$ as follows:

$$DH_g(h, j) = g^{(\log_g h) \cdot (\log_g j)}$$

- We note how:

$$DH_g(g^x, g^y) = g^{x \cdot y} = (g^x)^y = (g^y)^x.$$

- The CDH problem consists in efficiently computing DH_g. given a group $\mathbb{G}$ and a generator g for it (produced by GenCG).
- The assumption CDH (with respect to GenCG) holds when the problem CDH is hard (relative to GenCG).
- Every efficient algorithm for the discrete logarithm induces an efficient algorithm for CDH.
    - Just compute the logarithms of h and j, multiply the results and raise g to the product.

Decisional Diffie-Hellman Assumption

- Informally, the problem DDH consists in distinguishing $DH_g(h, j)$ from an arbitrary element of the group $\mathbb{G}$, given obviously h and j.

Decisional Diffie-Hellman Assumption

- Informally, the problem DDH consists in distinguishing $DH_g(h, j)$ from an arbitrary element of the group $\mathbb{G}$, given obviously h and j.
- Formally, we say that the problem DDH is hard (or that the assumption DDH is valid with respect to GenCG) iff for every PPT A there exists ε negligible such that

  $$|Pr(A(\mathbb{G}, q, g, g^x, g^y, g^z) = 1) - Pr(A(\mathbb{G}, q, g, g^x, g^y, g^{xy}))| \leq \varepsilon(n)$$

  where $(\mathbb{G}, q, g)$ is the result of $\mathsf{GenCG}(1^n)$ and $x, y, z \in \mathbb{Z}_q$ are random.
- Every efficient algorithm for CDH trivially induces an efficient algorithm for DDH.
  - A needs only to compute $DH_g(h, j)$ where h and j are the fourth and fifth parameter. The result must then be compared with the sixth parameter.

RSA Assumption $\longrightarrow$ Factoring Assumption

DDH Assumption $\longrightarrow$ CDH Assumption $\longrightarrow$ Discrete Logarithm Assumption

DH Assumptions on Specific Groups

- First of all it should be noted that the use of groups with a prime number of elements is to be preferred, this is because:
  - Testing whether an element is or is not a generator is trivial.
  - It is possible to show that if $\mathbb{G}$ is a group of prime order q with $|q| = \Theta(2^n)$ then

$$Pr(DH_g(h, j) = y) = \frac{1}{q} + \varepsilon(n)$$

- We then consider the groups $\mathbb{Z}_p^*$ where p is a *prime*.
  - An algorithm GenCG that generates groups of this type exists and it is efficient. The discrete logarithm assumption applies to this groups.
  - DDH is *not* believed to be hard for these groups.
  - However, there is a different algorithm GenCG' which returns a subset of $\mathbb{Z}_p^*$ and for which DDH is also believed to be hard.

From Factoring to One-Way Functions

- Consider a function $\mathsf{GenModulus}$ that takes as input at most $p(n)$ random bits of length n, where p is a polynomial.
- We construct an algorithm that computes a function $f_{\mathsf{GenModulus}}$ as follows:
  - The input is a string x;
  - Compute an integer n such that $p(n) \leq |x| \leq p(n+1)$;
  - Compute (N, p, q) as the result of $\mathsf{GenModulus}(1^n)$ using as random bits those in x, that are enough.
  - Return N.

From Factoring to One-Way Functions

- Consider a function $\mathsf{GenModulus}$ that takes as input at most $p(n)$ random bits of length n, where p is a polynomial.
- We construct an algorithm that computes a function $f_{\mathsf{GenModulus}}$ as follows:
  - The input is a string x;
  - Compute an integer n such that $p(n) \leq |x| \leq p(n+1)$;
  - Compute (N, p, q) as the result of $\mathsf{GenModulus}(1^n)$ using as random bits those in x, that are enough.
  - Return N.
- We observe now how the following distributions are identical for each $m \in \mathbb{N}$
  - The result N of $f_{\mathsf{GenModulus}}(x)$ where $x \in \{0,1\}^m$ is randomly chosen.
  - The result N of $\mathsf{GenModulus}(1^n)$ where $p(n) \leq m \leq p(n+1)$.

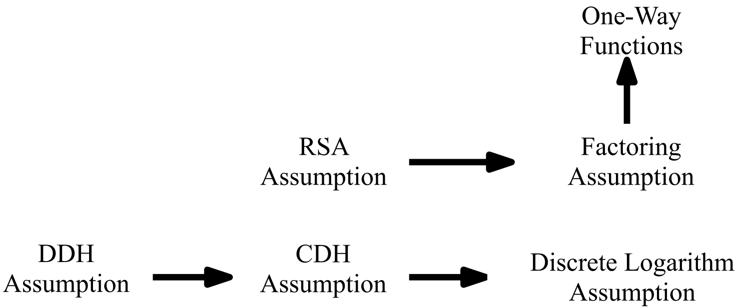
From Factoring to One-Way Functions

- Consider a function GenModulus that takes as input at most $p(n)$ random bits of length n, where p is a polynomial.
- We construct an algorithm that computes a function $f_{\mathsf{GenModulus}}$ as follows:
  - The input is a string x;
  - Compute an integer n such that $p(n) \leq |x| \leq p(n+1)$;
  - Compute (N, p, q) as the result of $\mathsf{GenModulus}(1^n)$ using as random bits those in x, that are enough.
  - Return N.
- We observe now how the following distributions are identical for each $m \in \mathbb{N}$
  - The result N of $f_{\mathsf{GenModulus}}(x)$ where $x \in \{0,1\}^m$ is randomly chosen.
  - The result N of $\mathsf{GenModulus}(1^n)$ where $p(n) \leq m \leq p(n+1)$.

Theorem

If factoring is hard relative to GenModulus, *then* $f_{\mathsf{GenModulus}}$ *is a one-way function.*

Summing Up


One-Way Functions
RSA Assumption
Factoring Assumption
DDH Assumption
CDH Assumption
Discrete Logarithm Assumption