Key Exchange and Public-key Encryption

6.1600 Course Staff

Fall 2023

So far, we have talked about encryption systems that require the sender and recipient to *share a secret key*. In this chapter, we discuss how the sender and recipient can agree on a shared secret even if they only ever communicate over an open (insecure) network.

Disclaimer: This set of notes is a work in progress. It may have errors and be missing citations. It is certainly incomplete. Please let the staff know of any errors that you find.

1 *Key exchange*

A key-agreement scheme over keyspace $\mathcal{K}$ is defined by efficient functions $(\mathsf{Gen}, \mathsf{Derive})$:

- $\mathsf{Gen}() \to (\mathsf{sk}, \mathsf{pk})$. The Gen algorithm generates a secret key and a public key for one party.
- $\mathsf{Derive}(\mathsf{sk}_A, \mathsf{pk}_B) \to k$. The Derive algorithm takes as input one party's secret key sk_A and the other party's public key pk_B and outputs a shared key $k \in \mathcal{K}$.

Formally, Gen also takes as input the security parameter.

Given this syntax, let us see how two parties can use a key-agreement scheme $(\mathsf{Gen}, \mathsf{Derive})$ to agree on a shared secret:

1. Alice runs $(\mathsf{sk}_A, \mathsf{pk}_A) \leftarrow \mathsf{Gen}()$ and sends pk_A to Bob.
2. Bob runs $(\mathsf{sk}_B, \mathsf{pk}_B) \leftarrow \mathsf{Gen}()$ and sends pk_B to Alice.
3. Alice and Bob both then compute a key $k \in \mathcal{K}$ as:
   - Alice computes $k \leftarrow \mathsf{Derive}(\mathsf{sk}_A, \mathsf{pk}_B)$.
   - Bob computes $k \leftarrow \mathsf{Derive}(\mathsf{sk}_B, \mathsf{pk}_A)$.

Correctness. Correctness for a key-agreement scheme just says that the two parties should always agree on the same shared secret:

Definition 1.1 (Key Agreement Correctness)**.** We say that a key-agreement scheme is *correct* if for all $(\mathsf{sk}_A, \mathsf{pk}_A) \leftarrow \mathsf{Gen}()$ and $(\mathsf{sk}_B, \mathsf{pk}_B) \leftarrow \mathsf{Gen}()$, it holds that:

$$\mathsf{Derive}(\mathsf{sk}_A, \mathsf{pk}_B) = \mathsf{Derive}(\mathsf{sk}_B, \mathsf{pk}_A).$$

Security. The standard notion of security for a key-agreement scheme only considers *passive attacks*: we consider adversaries that can view the network traffic but cannot modify it. In practice, we can combine key-agreement schemes with authentication schemes (e.g., digital signatures) to prevent active attacks by in-network adversaries.

Our security definition for key agreement says that even if an adversary sees both parties' public keys, it should not be able to distinguish the shared secret from random. That is, the following probability distributions D_{real} and D_{random} should be computationally indistinguishable:

When we say that two distributions are *computationally indistinguishable*, we mean that if we give the adversary a sample from one or the other, it can guess which sample it got with probability at most $1/2 +$ "negligible".

$$D_{\text{real}} := \left\{ (\mathsf{pk}_A, \mathsf{pk}_B, k) \colon \begin{array}{r} (\mathsf{sk}_A, \mathsf{pk}_A) \xleftarrow{R} \mathsf{Gen}() \\ (\mathsf{sk}_B, \mathsf{pk}_B) \xleftarrow{R} \mathsf{Gen}() \\ k \xleftarrow{R} \mathsf{Derive}(\mathsf{sk}_A, \mathsf{pk}_B) \end{array} \right\}$$

$$D_{\text{random}} := \left\{ (\mathsf{pk}_A, \mathsf{pk}_B, k) \colon \begin{array}{r} (\_, \mathsf{pk}_A) \xleftarrow{R} \mathsf{Gen}() \\ (\_, \mathsf{pk}_B) \xleftarrow{R} \mathsf{Gen}() \\ k \xleftarrow{R} \mathcal{K} \end{array} \right\}$$

2 *Diffie-Hellman key exchange*

We now give a simple and beautiful key-exchange protocol, due to Diffie and Hellman.

So far, we have been able to construct our cryptographic entities from constructs like a PRF, which meant that we could use "unstructured" algorithms like AES to compute them. We so far only know how to construct key-exchange schemes from more structured problems (e.g., based on number theory).

The version we will see uses large primes p and q a public parameters, along with a number $g \in \mathbb{Z}_p^*$, called the "generator." (**Warning:** The parameters p, q, and g must have some particular relation. So do not attempt to pick these parameters yourself.) Typically, we will have $p \approx q \approx 2^{2048}$.

For a prime p, the notation $\mathbb{Z}_p^*$ just denotes the non-zero integers modulo p. So when we write $ab \in \mathbb{Z}_p^*$, we mean $a \cdot b \bmod p$.

The keyspace of the Diffie-Hellman scheme is $\mathcal{K} = \mathbb{Z}_p^*$ and the algorithms are as follows:

- $\mathsf{Gen}() \to (\mathsf{sk}, \mathsf{pk})$.
  - Sample $\mathsf{sk} \xleftarrow{R} \{1, \dots, q\}$.
  - Set $\mathsf{pk} \leftarrow g^{\mathsf{sk}} \in \mathbb{Z}_p^*$.
  - Output $(\mathsf{sk}, \mathsf{pk})$.
- $\mathsf{Derive}(\mathsf{sk}_A, \mathsf{pk}_B) \to k$.
  - We have $\mathsf{sk}_A \in \{1, \dots, q\}$ and $\mathsf{pk}_B \in \mathbb{Z}_p^*$.
  - Output $k \leftarrow (\mathsf{pk}_B)^{\mathsf{sk}_A} \in \mathbb{Z}_p^*$.

Before we argue correctness and security, let us consider the computational efficiency of the scheme:

Efficiency. In order for the algorithm to be useful, Alice and Bob must be able to compute $g^x \in \mathbb{Z}_p^*$ efficiently, for $x \in \{1, \dots, q \approx 2^{2048}\}$. However, trying to compute g^x where x is 2048 bits long naively would certainly not be efficient: it would require $x \approx 2^{2048}$ multiplications! However, we can compute this exponentiation much more efficiently using the following strategy:

- **Compute powers of g.** Write $\ell \leftarrow \lceil \log_2 p \rceil$. Then compute $(g, g^2, g^4, g^8, g^{16}, \dots, g^{2^\ell})$, where all of these are in $\mathbb{Z}_p^*$. It is possible to compute g^{2^i} with a single multiplication modulo p as $g^{2^i} = (g^{2^{(i-1)}})^2 \in \mathbb{Z}_p^*$. So this step takes only $\ell = 2048$ multiplications.
- **Compute exponentiation.** Write the bits of the exponent as $x = x_0 \cdots x_{\ell-1}$. Then compute:
$$g^x = g^{\sum_{i=0}^{\ell-1} x_i 2^i} = \prod_{i=0}^{\ell-1} x_i (g^{2^i}) \in \mathbb{Z}_p^*$$
This step again takes only ℓ multiplications.

In many applications, the generator g is fixed in advance. In this case, the implementation can precompute and store these powers of g.

Correctness. Correctness holds since $g^{xy} = g^{yx} \in \mathbb{Z}_p^*$ for all $x, y \in \mathbb{Z}$:
$$\mathsf{Derive}(\mathsf{sk}_A, \mathsf{pk}_B) = (\mathsf{pk}_B)^{\mathsf{sk}_A} = (\mathsf{pk}_A)^{\mathsf{sk}_B} = \mathsf{Derive}(\mathsf{sk}_B, \mathsf{pk}_A).$$

Security. To argue security, we must rely on a new computational assumption: essentially we just assume that the key-agreement scheme is secure.

Definition 2.1 (Decision Diffie-Hellman (DDH) assumption)**.** The *decision Diffie-Hellman assumption* states that, for a suitable choice of p, q, and g, the following distributions are computationally indistinguishable:
$$D_{\text{real}} := \{(g, g^x, g^y, g^{xy}) \in (\mathbb{Z}_p^*)^4 : x, y \xleftarrow{R} \{1, \dots, q\}\}$$
$$D_{\text{ideal}} := \{(g, g^x, g^y, g^z) \in (\mathbb{Z}_p^*)^4 : x, y, z \xleftarrow{R} \{1, \dots, q\}\}$$

To make the statement fully formal, we need to let p, q, and g grow with the security parameter.

In practice, we typically first run Diffie-Hellman key agreement, have the two parties run the shared secret that they get through a cryptographic hash function, and then use the hashed value as an encryption key. If we model the hash function as a random oracle, security can rely on a slightly weaker assumption—the "computational" Diffie-Hellman (CDH) assumption. The CDH assumption asserts, informally, that given (g, g^x, g^y), it is infeasible to compute g^{xy}. More formally, we have:

Definition 2.2 (Computational Diffie-Hellman (CDH) assumption)**.** The *computational Diffie-Hellman assumption* states that, for a suitable choice of p, q, and g, and for all adversaries $\mathcal{A}$:
$$\Pr[\mathcal{A}(g, g^x, g^y) = g^{xy} : x, y, \xleftarrow{R} \{1, \dots, q\}] \leq \text{"negligible."}$$

3 *The discrete-log problem*

The DDH assumption (Definition 2.1) is no harder than the following problem, which asserts that computing x given $(g, g^x) \in (\mathbb{Z}_p^*)^2$ is computationally infeasible:

Definition 3.1 (Discrete-log assumption)**.** The *discrete-log assumption* states that, for p, q, and g, and for all efficient adversaries $\mathcal{A}$:

$$\Pr[\mathcal{A}(g, g^x) = x \colon x \xleftarrow{R} \{1, \dots, q\}] \leq \text{“negligible”}.$$

Given an algorithm for the discrete-log problem, we can use it to solve the DDH problem. Given a DDH challenge (g, g^x, g^y, g^z), compute the discrete log of g^x and test whether $g^z = (g^y)^x$. For certain choices of p, q, and g, the best known algorithm for the DDH problem is to first solve the discrete-log problem in $\mathbb{Z}_p^*$.

How hard it to solve the discrete-log problem then?

1. The most basic attack is to enumerate all p possible values of x and check whether the corresponding g^x matches. This will take time $p \approx 2^{2048}$.

2. There is a slightly more clever algorithm, called “Baby Step Giant Step,” that is able to compute x in time $\sqrt{p}$.

3. In $\mathbb{Z}_p^*$, a much better attack is the Number Field Sieve. This algorithm is able to compute x in (roughly) time $\exp((\log p)^{1/3} (\log \log p)^{2/3})$—sub-exponential time! The existence of this attack is the reason we require p to be 2048 bits long to get 128-bit security.

An attack that runs in time $\sqrt{p}$ runs in time $2^{\frac{1}{2}\log p}$. In contrast, the Number Field Sieve runs in time roughly $2^{\sqrt[3]{\log p}}$. This is much much much faster than the $\sqrt{p}$-time algorithms, since the exponent grows much more slowly.

4 *Generalizations of Diffie-Hellman*

We have described the Diffie-Hellman protocol in terms of $\mathbb{Z}_p^*$—multiplication of integers modulo p. In particular, the protocol uses a *set* of elements (here, $\mathbb{Z}_p^*$) and a *binary operation on elements* (here, multiplication modulo p). There is a natural generalization of the Diffie-Hellman protocol that works with other sets of elements and binary operations that operate on them.

More precisely, we can define Diffie-Hellman key exchange with respect to any *finite cyclic group*—a concept from abstract algebra. A cyclic group just consists of a *set* of elements and a *binary operation on elements*. The set and operation need to satisfy certain mathematical properties—associativity, etc.

Given a group $\mathbb{G}$, we can then define a discrete-log assumption on $\mathbb{G}$ and whenever discrete-log is hard on $\mathbb{G}$, we can use $\mathbb{G}$ to construct cryptosystems.

The most widely used version of the Diffie-Hellman protocol today uses *elliptic-curve groups.* The set of elements in an elliptic-curve group is a set of points $(x, y) \in \mathbb{Z}^2$ in two-dimensional space, where $0 \leq x, y \leq p$ for some prime $p \approx 2^{256}$. The binary operation on elements is some geometric operation on two points that yields a third point. Even though the underlying objects are now points instead of integers modulo p, the Diffie-Hellman protocol looks exactly the same in this setting.

The appeal of elliptic-curve cryptosystems is that the best known discrete-log algorithm on certain elliptic-curve groups is Baby Step Giant Step, which runs in $\sqrt{p}$ time. So, we can use elliptic-curve groups of size 2^{256} and the Diffie-Hellman public keys take only ≈ 256 bits to represent. In contrast, when working in $\mathbb{Z}_p^*$, we need to work modulo a prime $p \approx 2^{2048}$ to defeat the Number Field Sieve attack, so Diffie-Hellman public keys in this setting take ≈ 2048 bits to represent.

5 *Defining Public-Key Encryption*

The definition for a public-key encryption scheme will be similar to the definitions we saw for symmetric-key encryption:

Definition 5.1 (Public-Key Encryption Scheme)**.** A public-key encryption scheme over message space $\mathcal{M}$ consists of three efficient algorithms $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$:

- $\mathsf{Gen}() \to (\mathsf{sk}, \mathsf{pk})$: Generates a keypair with secret key sk and public key pk.
- $\mathsf{Enc}(\mathsf{pk}, m) \to c$: Uses public key pk to encrypts a message $m \in \mathcal{M}$ to a ciphertext c.
- $\mathsf{Dec}(\mathsf{sk}, c) \to m$: Uses secret key sk to decrypt ciphertext c into message $m \in \mathcal{M}$.

Definition 5.2 (Public-Key Encryption - Correctness)**.** For all keypairs $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}()$ and for all messages $m \in \mathcal{M}$, it holds that

$$\mathsf{Dec}(\mathsf{sk}, \mathsf{Enc}(\mathsf{pk}, m)) = m.$$

The literature sometimes calls security against chosen-plaintext attacks ("CPA security") *semantic security*.

Definition 5.3 (Public-Key Encryption - Security against chosen-plaintext attacks (Weak))**.** A public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is *secure against chosen-plaintext attacks* if all efficient adversaries $\mathcal{A}$ win the following game with probability $\leq \frac{1}{2} + \text{negl}$:

- The challenger generates $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}()$ and $b \xleftarrow{R} \{0,1\}$ and sends the public key pk to $\mathcal{A}$.
- The adversary $\mathcal{A}$ sends $m_0, m_1 \in \mathcal{M}$ to the challenger, where $|m_0| = |m_1|$.
- The challenger responds with $\mathsf{Enc}(\mathsf{pk}, m_b)$
- The adversary outputs b' and wins if $b' = b$.

Note that since this is a public-key encryption scheme, the adversary can use the public key to generate encryptions of any message of their choice. As in the secret-key setting, here it is crucial that the encryption algorithm be randomized—otherwise two encryptions of the same message are identical and the attacker can break CPA security.

For symmetric-key encryption, we also defined a stronger notion of security that we called security against chosen-ciphertext attacks (CCA security). We can extend this definition of CCA security to the public-key setting.

This security definition asserts that the attacker should not be able to distinguish the encryption c^* of two messages of its choice, even if the attacker can obtain decryptions of any ciphertext except c^*. This is a very strong notion of security (since the security definition gives the attacker a lot of power) and it is the notion of security that we typically demand in practice.

Definition 5.4 (Public-Key Encryption - Security against chosen–ciphertext attacks (Strong))**.** A public-key encryption scheme $(\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec})$ is *secure against chosen-ciphertext attacks* if all efficient adversaries $\mathcal{A}$ win the following game with probability $\leq \frac{1}{2} + \text{negl}$:

- The challenger generates $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{Gen}()$ and $b \overset{\mathrm{R}}{\leftarrow} \{0,1\}$ and sends the public key pk to the adversary $\mathcal{A}$.
- The adversary may make polynomially many *decryption queries*:
  - The adversary $\mathcal{A}$ sends ciphertext c to the challenger.
  - The challenger responds with $m \leftarrow \mathsf{Dec}(\mathsf{sk}, c)$
- At some point, the adversary sends a pair of messages $m_0, m_1 \in \mathcal{M}$ to the challenger, where $|m_0| = |m_1|$.
- The challenger returns $c^* \leftarrow \mathsf{Enc}(\mathsf{pk}, m_b)$.
- The adversary can continue to make decryption queries, provided that it never asks the challenger to decrypt the ciphertext c^*.
- The adversary outputs b' and wins if $b' = b$.

6 *ElGamal Encryption Scheme*

In public-key encryption, we want a sender to be able to encrypt a message to a recipient. The goal of public-key encryption is to perform encryption without a shared secret key.

ElGamal's encryption scheme essentially uses Diffie-Hellman key exchange to allow the sender and recipient to agree on a shared secret key k, and then has the sender encrypt her message with a symmetric-key cryptosystem under key k.

ElGamal's scheme was not the first public-key encryption scheme. The first scheme, RSA, was much more complicated despite Diffie-Hellman already having been published.

In practice, we typically use elliptic-curve groups to instantiate ElGamal encryption, instead of $\mathbb{Z}_p^*$. But the general principle is exactly the same.

Definition 6.1 (Hashed ElGamal Encryption)**.** Let p, q, and g be integers of the sort we use for Diffie-Hellman key exchange (Section 2). In particular, $p \approx q \approx 2^{2048}$ and $g \in \mathbb{Z}_p^*$.

Let $H : \mathbb{Z}_p^* \to \{0,1\}^*$ be a hash function (modelled as a random oracle). Let $(\mathsf{Enc}', \mathsf{Dec}')$ be a symmetric-key authenticated-encryption scheme. Then define:

- $\mathsf{Gen}() \to (\mathsf{sk}, \mathsf{pk})$:
  - Choose $a \leftarrow \{1, \ldots, q\}$.
  - Compute $A \leftarrow g^a \in \mathbb{Z}_p^*$.
  - Output $(\mathsf{sk}, \mathsf{pk}) \leftarrow (a, A)$.
- $\mathsf{Enc}(\mathsf{pk}, m) \to c$:
  - Choose $r \leftarrow \{1, \ldots, q\}$.
  - Compute $R \leftarrow g^r \in \mathbb{Z}_p^*$.

  - Compute $k \leftarrow H(\mathsf{pk}^r \in \mathbb{Z}_p^*)$.
  - Output $c \leftarrow (R, \mathsf{Enc}'(k, m)$.

- $\mathsf{Dec}(\mathsf{sk}, c) \to m$:
  - Parse $(R, c') \leftarrow c$.
  - Compute $k \leftarrow H(R^a \in \mathbb{Z}_p^*)$.
  - Output $m \leftarrow \mathsf{Dec}'(k, c')$.

6.1 Performance

The performance of this scheme is limited by the exponentiations—the symmetric encryption scheme is quite fast (gigabytes per second), but a single exponentiation can take a millisecond on modern processors. For encryption, this scheme requires two exponentiations (g^r and pk^r). Decryption requires one exponentiation (R^{sk}). To speed up the encryption routine, we can precompute powers of g: $g^2, g^4, g^8, g^{16}, \dots$, which saves a factor of two in exponentiations.

Hashed ElGamal encryption is one of the most common public-key encryption schemes used today.

6.2 Security

If we model the hash function H as a random oracle, we can prove security of ElGamal encryption from (a) the computational Diffie-Hellman assumption (Definition 2.2) and (b) the CCA security of the underlying authenticated-encryption scheme $(\mathsf{Enc}', \mathsf{Dec}')$.