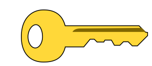
The Key Exchange Experiment

Fix a key exchange protocol Π and an attacker $\mathcal{A}$

We define the **key-exchange experiment** $\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ as follows:

- The honest parties run Π using n as the security parameter.
- The interaction results in a transcript τ and in a shared key $k \in \{0,1\}^n$

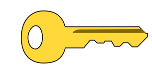


The Key Exchange Experiment

Fix a key exchange protocol Π and an attacker $\mathcal{A}$

We define the **key-exchange experiment** $\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ as follows:

- The honest parties run Π using n as the security parameter.
- The interaction results in a transcript τ and in a shared key $k \in \{0,1\}^n$



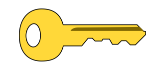
- A random bit b is chosen u.a.r. from $\{0,1\}$.
  - If $b = 0$ then $k' \leftarrow k$.
  - Otherwise (when $b = 1$) k' is chosen as a random uniform string from $\{0,1\}^n$.

The Key Exchange Experiment

Fix a key exchange protocol Π and an attacker $\mathcal{A}$

We define the **key-exchange experiment** $\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ as follows:

- The honest parties run Π using n as the security parameter.
- The interaction results in a transcript τ and in a shared key $k \in \{0,1\}^n$



- A random bit b is chosen u.a.r. from $\{0,1\}$.
    - If $b = 0$ then $k' \leftarrow k$.
    - Otherwise (when $b = 1$) k' is chosen as a random uniform string from $\{0,1\}^n$.
- $\mathcal{A}$ is given k' and the transcript τ

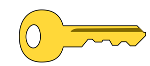
- $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$

The Key Exchange Experiment

Fix a key exchange protocol Π and an attacker $\mathcal{A}$

We define the **key-exchange experiment** $\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ as follows:

- The honest parties run Π using n as the security parameter.
- The interaction results in a transcript τ and in a shared key $k \in \{0,1\}^n$

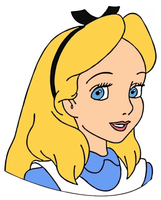


- A random bit b is chosen u.a.r. from $\{0,1\}$.
  - If $b = 0$ then $k' \leftarrow k$.
  - Otherwise (when $b = 1$) k' is chosen as a random uniform string from $\{0,1\}^n$.
- $\mathcal{A}$ is given k' and the transcript τ

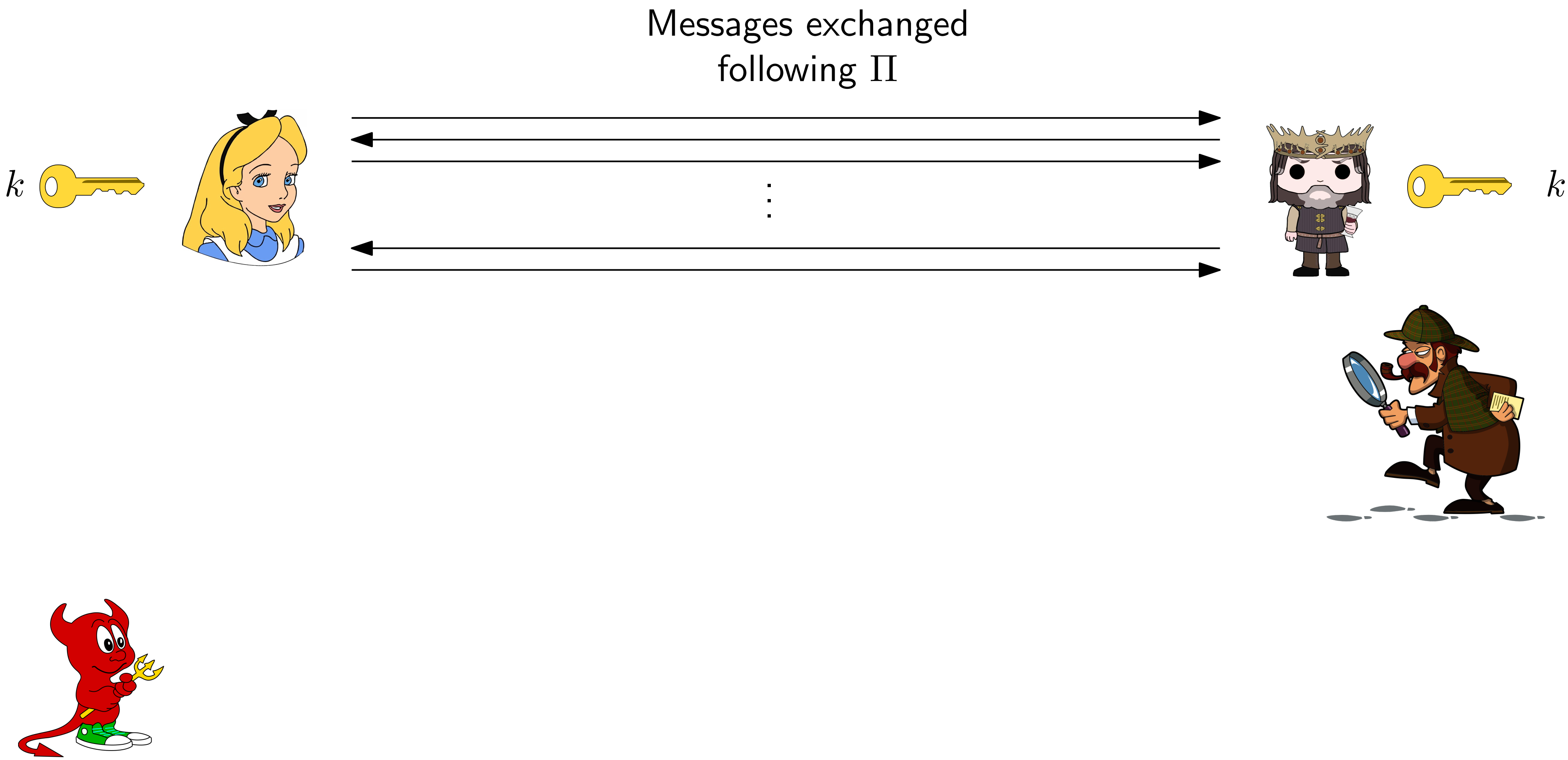
- $\mathcal{A}$ outputs a bit $b' \in \{0,1\}$

The outcome of the experiment is defined to be 1 if $b' = b$ and 0 otherwise

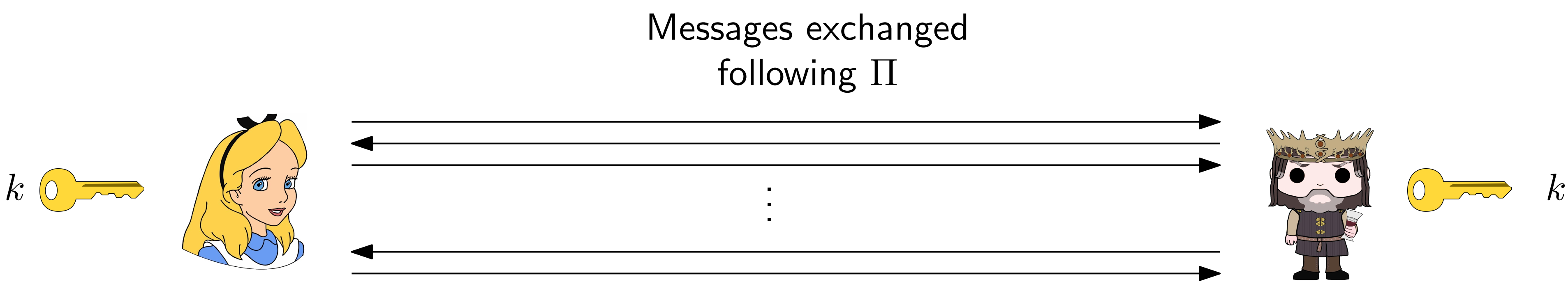
The Key Exchange Experiment



The Key Exchange Experiment



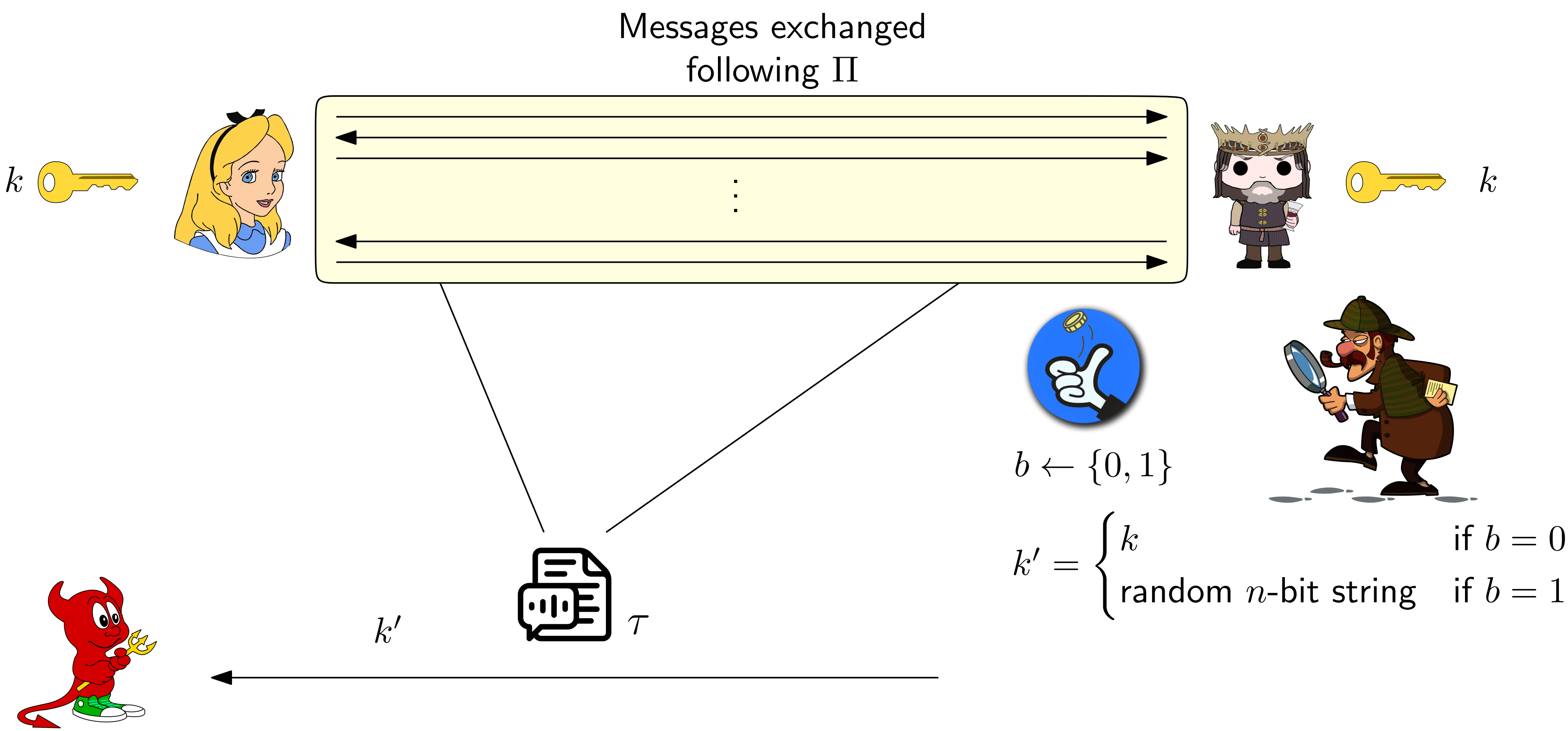
The Key Exchange Experiment



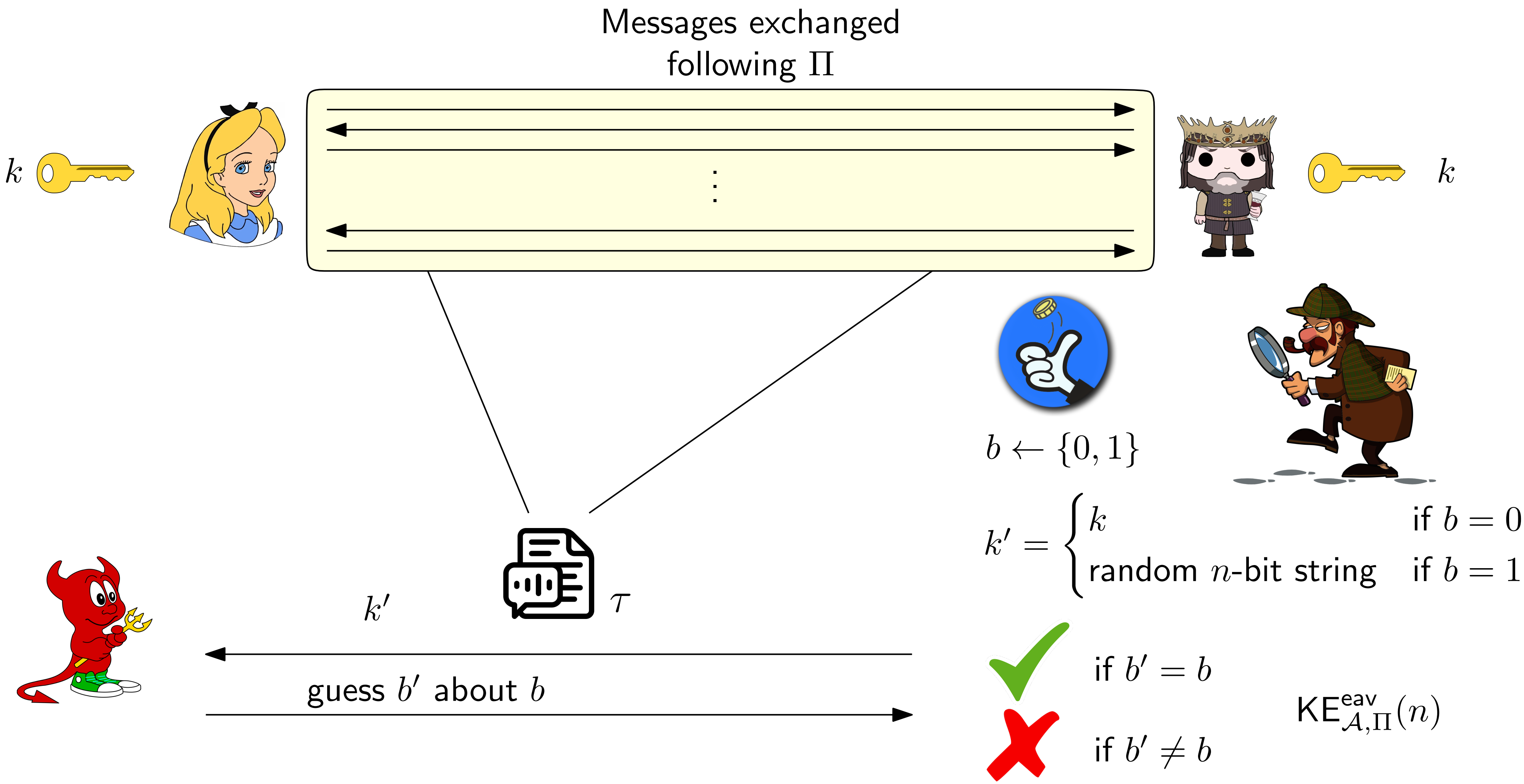
$b \leftarrow \{0,1\}$

$$k' = \begin{cases} k & \text{if } b = 0 \\ \text{random } n\text{-bit string} & \text{if } b = 1 \end{cases}$$

The Key Exchange Experiment



The Key Exchange Experiment



Key Exchange: Formal Security Definition

Definition: A key-exchange protocol Π is **secure in the presence of an eavesdropper** if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function ε such that

$$\Pr[\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \varepsilon(n).$$

Key Exchange: Formal Security Definition

Definition: A key-exchange protocol Π is **secure in the presence of an eavesdropper** if for all probabilistic polynomial-time adversaries $\mathcal{A}$ there is a negligible function ε such that

$$\Pr[\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \varepsilon(n).$$

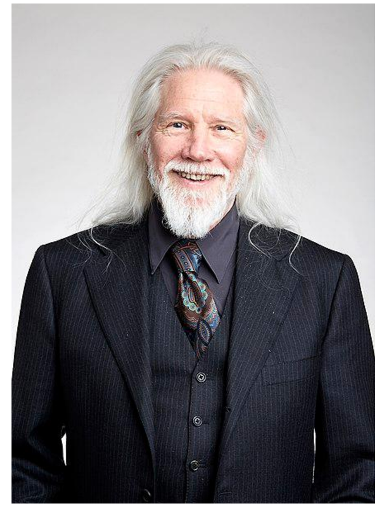
Notice that being unable to compute k from the transcript τ is not a strong enough security guarantee

- The requirement we impose is **stronger**. Namely k must look just like a random string.
- This is necessary since we are going to use k for private-key cryptography.

Diffie-Hellman Key Exchange

Let $\mathcal{G}$ be a probabilistic polynomial-time algorithm that, on input 1^n , outputs a description of a cyclic group G, its order q, where q is a n-bit integer, and a generator $g \in G$.

- Alice runs $\mathcal{G}(1^n)$ to obtain (G, q, g).



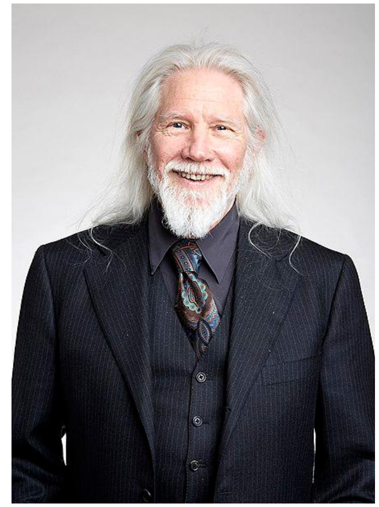
Whitfield Diffie

Martin Hellman

Diffie-Hellman Key Exchange

Let $\mathcal{G}$ be a probabilistic polynomial-time algorithm that, on input 1^n , outputs a description of a cyclic group G, its order q, where q is a n-bit integer, and a generator $g \in G$.

- Alice runs $\mathcal{G}(1^n)$ to obtain (G, q, g).
- Alice chooses a uniform $x \in \{0, 1, \ldots, q-1\}$ and computes $h_A = g^x$



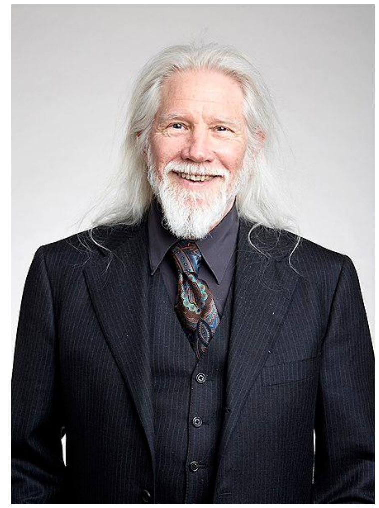
Whitfield Diffie

Martin Hellman

Diffie-Hellman Key Exchange

Let $\mathcal{G}$ be a probabilistic polynomial-time algorithm that, on input 1^n , outputs a description of a cyclic group G, its order q, where q is a n-bit integer, and a generator $g \in G$.

- Alice runs $\mathcal{G}(1^n)$ to obtain (G, q, g).
- Alice chooses a uniform $x \in \{0, 1, \ldots, q-1\}$ and computes $h_A = g^x$
- Alice sends (G, q, g, h_A) to Bob



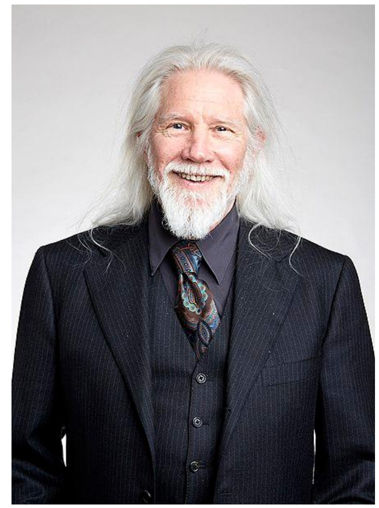
Whitfield Diffie

Martin Hellman

Diffie-Hellman Key Exchange

Let $\mathcal{G}$ be a probabilistic polynomial-time algorithm that, on input 1^n , outputs a description of a cyclic group G, its order q, where q is a n-bit integer, and a generator $g \in G$.

- Alice runs $\mathcal{G}(1^n)$ to obtain (G, q, g).
- Alice chooses a uniform $x \in \{0, 1, \ldots, q-1\}$ and computes $h_A = g^x$
- Alice sends (G, q, g, h_A) to Bob
- Bob receives (G, q, g, h_A), chooses a uniform $y \in \{0, 1, \ldots, q-1\}$ and computes $h_B = g^y$.



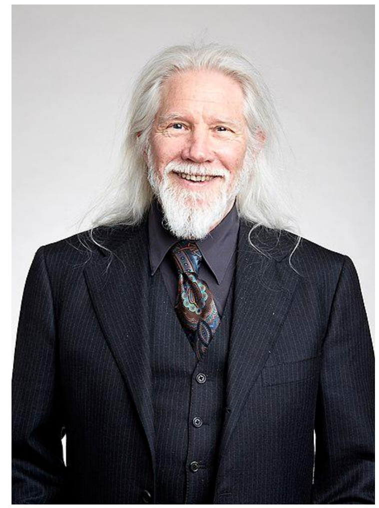
Whitfield Diffie

Martin Hellman

Diffie-Hellman Key Exchange

Let $\mathcal{G}$ be a probabilistic polynomial-time algorithm that, on input 1^n , outputs a description of a cyclic group G, its order q, where q is a n-bit integer, and a generator $g \in G$.

- Alice runs $\mathcal{G}(1^n)$ to obtain (G, q, g).
- Alice chooses a uniform $x \in \{0, 1, \ldots, q-1\}$ and computes $h_A = g^x$
- Alice sends (G, q, g, h_A) to Bob
- Bob receives (G, q, g, h_A), chooses a uniform $y \in \{0, 1, \ldots, q-1\}$ and computes $h_B = g^y$.
- Bob sends h_B to Alice and outputs $k = h_A^y$



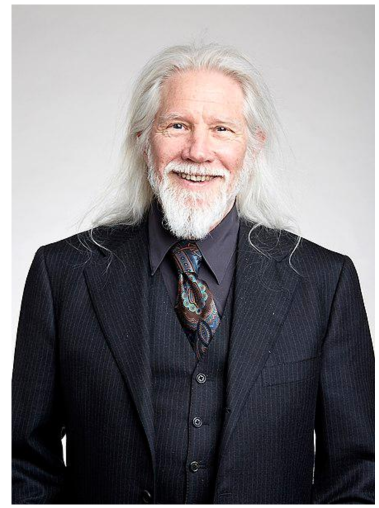
Whitfield Diffie

Martin Hellman

Diffie-Hellman Key Exchange

Let $\mathcal{G}$ be a probabilistic polynomial-time algorithm that, on input 1^n , outputs a description of a cyclic group G, its order q, where q is a n-bit integer, and a generator $g \in G$.

- Alice runs $\mathcal{G}(1^n)$ to obtain (G, q, g).
- Alice chooses a uniform $x \in \{0, 1, \ldots, q-1\}$ and computes $h_A = g^x$
- Alice sends (G, q, g, h_A) to Bob
- Bob receives (G, q, g, h_A), chooses a uniform $y \in \{0, 1, \ldots, q-1\}$ and computes $h_B = g^y$.
- Bob sends h_B to Alice and outputs $k = h_A^y$
- Alice receives h_B from Bob and outputs $k = h_B^x$



Whitfield Diffie

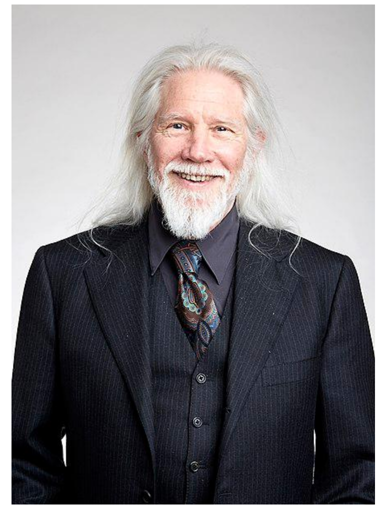
Martin Hellman

Diffie-Hellman Key Exchange

Let $\mathcal{G}$ be a probabilistic polynomial-time algorithm that, on input 1^n , outputs a description of a cyclic group G, its order q, where q is a n-bit integer, and a generator $g \in G$.

- Alice runs $\mathcal{G}(1^n)$ to obtain (G, q, g).
- Alice chooses a uniform $x \in \{0, 1, \ldots, q-1\}$ and computes $h_A = g^x$
- Alice sends (G, q, g, h_A) to Bob
- Bob receives (G, q, g, h_A), chooses a uniform $y \in \{0, 1, \ldots, q-1\}$ and computes $h_B = g^y$.
- Bob sends h_B to Alice and outputs $k = h_A^y$
- Alice receives h_B from Bob and outputs $k = h_B^x$

Notice that $k = h_A^y = (g^x)^y = g^{xy} = (g^y)^x = h_B^x$



Whitfield Diffie

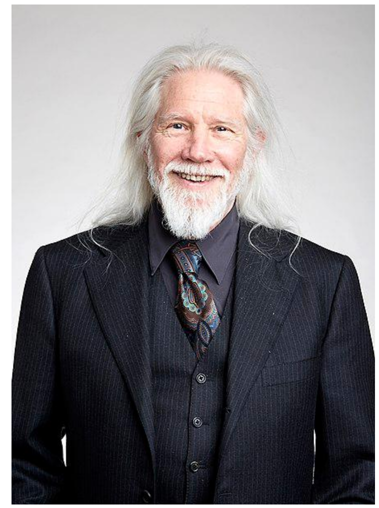
Martin Hellman

Diffie-Hellman Key Exchange

Let $\mathcal{G}$ be a probabilistic polynomial-time algorithm that, on input 1^n , outputs a description of a cyclic group G, its order q, where q is a n-bit integer, and a generator $g \in G$.

- Alice runs $\mathcal{G}(1^n)$ to obtain (G, q, g).
- Alice chooses a uniform $x \in \{0, 1, \ldots, q-1\}$ and computes $h_A = g^x$
- Alice sends (G, q, g, h_A) to Bob
- Bob receives (G, q, g, h_A), chooses a uniform $y \in \{0, 1, \ldots, q-1\}$ and computes $h_B = g^y$.
- Bob sends h_B to Alice and outputs $k = h_A^y$
- Alice receives h_B from Bob and outputs $k = h_B^x$

Notice that $k = h_A^y = (g^x)^y = g^{xy} = (g^y)^x = h_B^x$



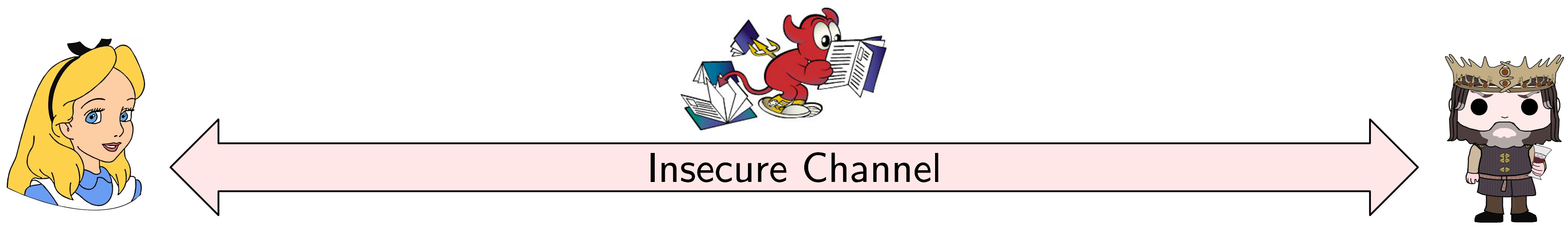
Whitfield Diffie

Martin Hellman

In practice the group G and the generator g are fixed and already known by Alice and Bob

Alice only needs to send h^A to Bob

Diffie-Hellman Key Exchange

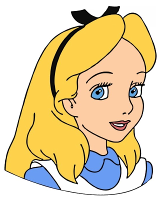


Suppose that Alice and Bob agreed to use a group G of order q and a generator $g \in G$.

- Pick x u.a.r. from $\{0, \ldots, q-1\}$

- Pick y u.a.r. from $\{0, \ldots, q-1\}$

Diffie-Hellman Key Exchange



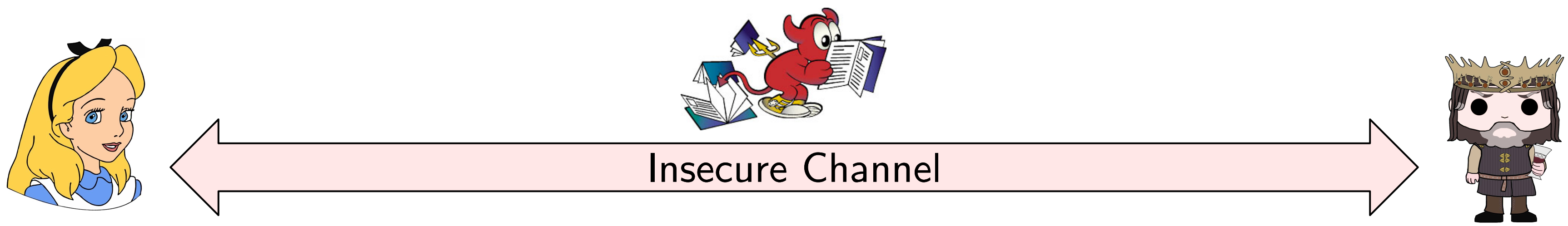
Insecure Channel

Suppose that Alice and Bob agreed to use a group G of order q and a generator $g \in G$.

- Pick x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_A = g^x$

- Pick y u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_B = g^y$

Diffie-Hellman Key Exchange

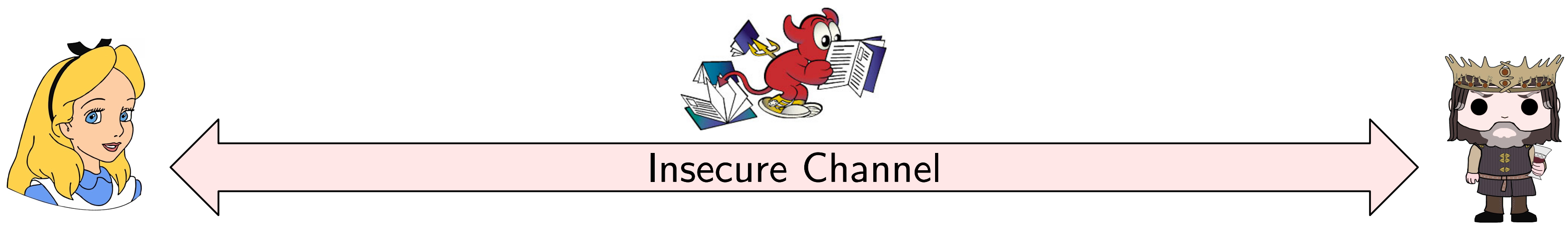


Suppose that Alice and Bob agreed to use a group G of order q and a generator $g \in G$.

- Pick x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_A = g^x$
- Send h_A to Bob

- Pick y u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_B = g^y$
- Send h_B to Alice

Diffie-Hellman Key Exchange

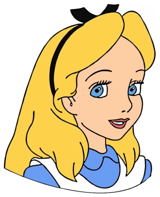


Suppose that Alice and Bob agreed to use a group G of order q and a generator $g \in G$.

- Pick x u.a.r. from $\{0, \dots, q-1\}$
- Compute $h_A = g^x$
- Send h_A to Bob
- Compute $h_B^x = (g^y)^x = g^{xy}$

- Pick y u.a.r. from $\{0, \dots, q-1\}$
- Compute $h_B = g^y$
- Send h_B to Alice
- Compute $h_A^y = (g^x)^y = g^{xy}$

Diffie-Hellman Key Exchange

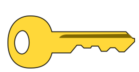


Insecure Channel

Suppose that Alice and Bob agreed to use a group G of order q and a generator $g \in G$.

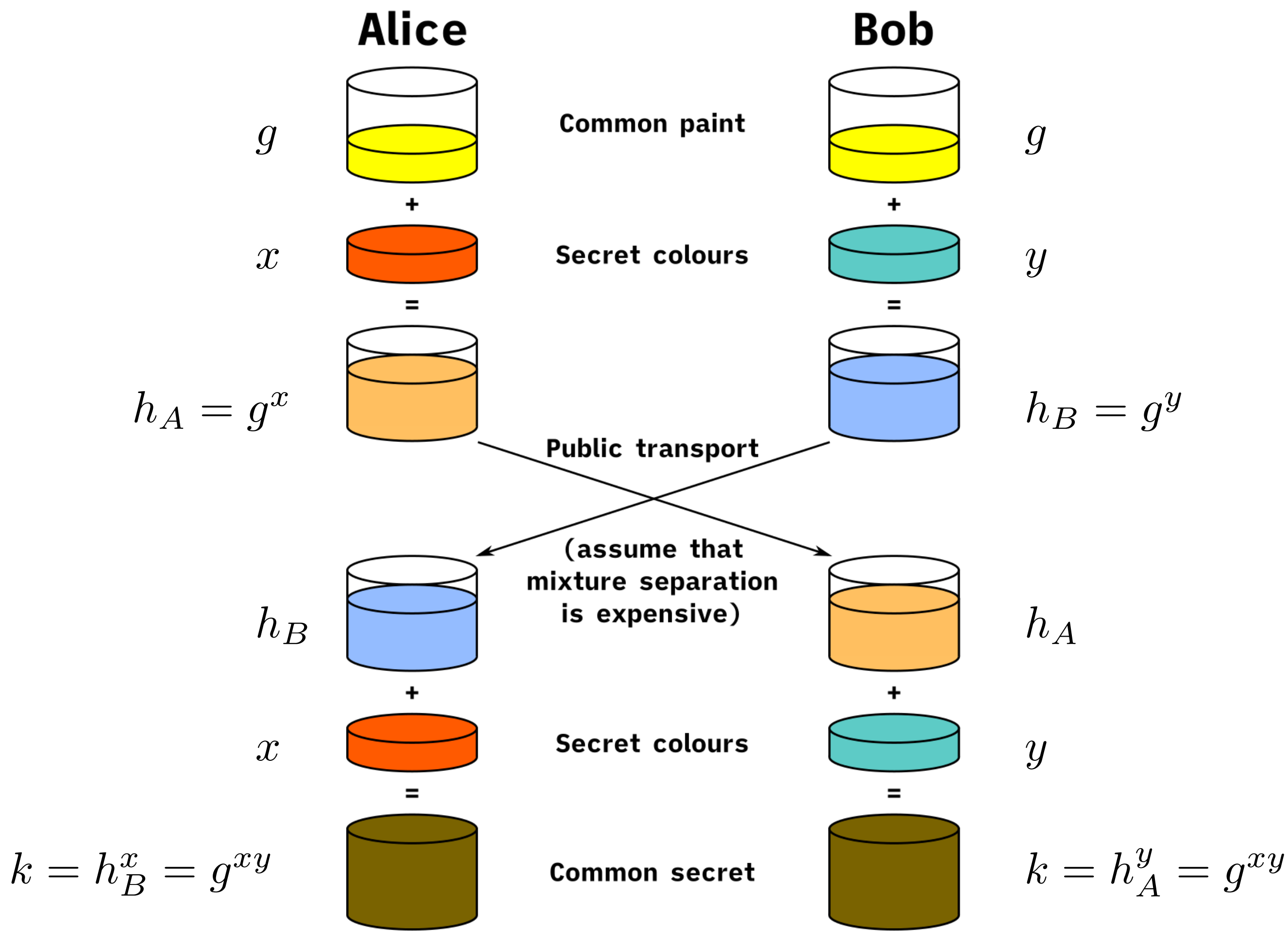
- Pick x u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_A = g^x$
- Send h_A to Bob
- Compute $h_B^x = (g^y)^x = \boxed{g^{xy}}$

- Pick y u.a.r. from $\{0, \ldots, q-1\}$
- Compute $h_B = g^y$
- Send h_B to Alice
- Compute $h_A^y = (g^x)^y = \boxed{g^{xy}}$



The shared secret key is $k = g^{xy}$

Diffie-Hellman Key Exchange



Diffie-Hellman Key Exchange: Example

Suppose that Alice and Bob agreed to use the group $G = \mathbb{Z}_{11}^*$ (of order $q = 10$) and the generator $g = 2$

This is just for the sake of the example! Recall that Z_{11}^* is **not** a good choice!

- Alice picks $x = 4$
  (chosen u.a.r. from $\{0, \ldots, 10\}$)

- Bob picks $y = 3$
  (chosen u.a.r. from $\{0, \ldots, 10\}$)

Diffie-Hellman Key Exchange: Example

Suppose that Alice and Bob agreed to use the group $G = \mathbb{Z}_{11}^*$ (of order $q = 10$) and the generator $g = 2$

This is just for the sake of the example! Recall that Z_{11}^* is **not** a good choice!

- Alice picks $x = 4$
  (chosen u.a.r. from $\{0, \ldots, 10\}$)
- Alice computes $h_A = 2^4$

- Bob picks $y = 3$
  (chosen u.a.r. from $\{0, \ldots, 10\}$)
- Bob computes $h_B = 2^3$

Diffie-Hellman Key Exchange: Example

Suppose that Alice and Bob agreed to use the group $G = \mathbb{Z}_{11}^*$ (of order $q = 10$) and the generator $g = 2$

This is just for the sake of the example! Recall that Z_{11}^* is **not** a good choice!

- Alice picks $x = 4$
  (chosen u.a.r. from $\{0, \dots, 10\}$)
- Alice computes $h_A = 2^4 = 5$

- Bob picks $y = 3$
  (chosen u.a.r. from $\{0, \dots, 10\}$)
- Bob computes $h_B = 2^3 = 8$

Diffie-Hellman Key Exchange: Example

Suppose that Alice and Bob agreed to use the group $G = \mathbb{Z}_{11}^*$ (of order $q = 10$) and the generator $g = 2$

This is just for the sake of the example! Recall that Z_{11}^* is **not** a good choice!

- Alice picks $x = 4$
  (chosen u.a.r. from $\{0, \ldots, 10\}$)

- Bob picks $y = 3$
  (chosen u.a.r. from $\{0, \ldots, 10\}$)

- Alice computes $h_A = 2^4 = 5$

- Bob computes $h_B = 2^3 = 8$

- Alice sends h_A to Bob

h_A

h_B

- Bob sends h_B to Alice

Diffie-Hellman Key Exchange: Example

Suppose that Alice and Bob agreed to use the group $G = \mathbb{Z}_{11}^*$ (of order $q = 10$) and the generator $g = 2$

This is just for the sake of the example! Recall that Z_{11}^* is **not** a good choice!

• Alice picks $x = 4$ (chosen u.a.r. from $\{0, \ldots, 10\}$)		• Bob picks $y = 3$ (chosen u.a.r. from $\{0, \ldots, 10\}$)
• Alice computes $h_A = 2^4 = 5$		• Bob computes $h_B = 2^3 = 8$
• Alice sends h_A to Bob	$\xrightarrow{h_A}$	
	$\xleftarrow{h_B}$	• Bob sends h_B to Alice
• Alice computes $k = 8^4 = 4$		• Bob computes $k = 5^3 = 4$

The shared secret key is $k = 4$

Diffie-Hellman Key Exchange: Example

Suppose that Alice and Bob agreed to use the group $G = \mathbb{Z}_{11}^*$ (of order $q = 10$) and the generator $g = 2$

This is just for the sake of the example! Recall that Z_{11}^* is **not** a good choice!

- Alice picks $x = 4$
  (chosen u.a.r. from $\{0, \ldots, 10\}$)
- Alice computes $h_A = 2^4 = 5$
- Alice sends h_A to Bob

- Bob picks $y = 3$
  (chosen u.a.r. from $\{0, \ldots, 10\}$)
- Bob computes $h_B = 2^3 = 8$
- Bob sends h_B to Alice

$\xrightarrow{h_A}$

$\xleftarrow{h_B}$

- Alice computes $k = 8^4 = 4$
- Bob computes $k = 5^3 = 4$

The shared secret key is $k = 4$

Diffie-Hellman Key Exchange: Security

The adversary sees (or knows already) G, q, g

The adversary sees $h_A = g^x$ and $h_B = g^y$

The shared key is $k = g^{xy}$

What kind of assumption do we need to guarantee security of the Diffie-Hellman Key Exchange?

Diffie-Hellman Key Exchange: Security

The adversary sees (or knows already) G, q, g

The adversary sees $h_A = g^x$ and $h_B = g^y$

The shared key is $k = g^{xy}$

What kind of assumption do we need to guarantee security of the Diffie-Hellman Key Exchange?

Wait... what were the assumptions again?

A reminder...

Reminder: The Discrete Logarithm Assumption

For a group-generation algorithm $\mathcal{G}$ and an algorithm $\mathcal{A}$, define the experiment $\mathsf{DLog}_{\mathcal{A},\mathcal{G}}(n)$ as:

- Run $\mathcal{G}(1^n)$ to obtain (G, q, g), where G is a cyclic group of order q (where q is a n-bit integer), and g is a generator of G.
- Choose a uniform $h \in G$.
- G, q, g and h are given to $\mathcal{A}$
- $\mathcal{A}$ outputs $x \in \{0, \ldots q-1\}$
- The outcome of the experiment is 1 if $g^x = h$. Otherwise the outcome is 0.

Definition The discrete-logarithm problem is hard relative to $\mathcal{G}$ if for all probabilistic polynomial-time algorithms $\mathcal{A}$ there exists a negligible function ε

$$\Pr[\mathsf{DLog}_{\mathcal{A},\mathcal{G}}(n) = 1] \leq \varepsilon(n).$$

The discrete logarithm assumption: there exists a group-generation algorithm $\mathcal{G}$ for which the discrete-logarithm problem is hard

Reminder: The CHD Problem and Assumption

Given $g, h_1, h_2 \in G$, define: $\mathsf{DH}_g(h_1, h_2) = g^{\log_g h_1 \cdot \log_g h_2}$

The **Computational Diffie-Hellman (CDH) problem** is that of computing $\mathsf{DH}_g(h_1, h_2)$ given a group G, a generator g, and two elements h_1, and h_2 chosen u.a.r. from G

Definition The CDH problem is hard relative to $\mathcal{G}$ if for all probabilistic polynomial-time algorithms $\mathcal{A}$ there exists a negligible function ε such that

$$\Pr[\mathcal{A}(G, q, g, h_1, h_2) = \mathsf{DH}_g(h_1, h_2)] = \varepsilon(n),$$

where the probabilities are taken over the experiment in which $\mathcal{G}(1^n)$ outputs (G, q, g), and uniform $h_1, h_2 \in G$ are chosen.

The CDH assumption: there exists a group-generation algorithm $\mathcal{G}$ for which the CDH problem is hard

Reminder: the DDH Problem and Assumption

Definition The DDH problem is hard relative to $\mathcal{G}$ if for all probabilistic polynomial-time algorithms $\mathcal{A}$ there exists a negligible function ε such that

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

where the probabilities are taken over the experiment in which $\mathcal{G}(1^n)$ outputs (G, q, g), and then uniform $x, y, z \in \{0, 1, \ldots, q-1\}$ are chosen (therefore g^x and g^y are uniformly distributed in G).

The DDH assumption: there exists a group-generation algorithm $\mathcal{G}$ for which the DDH problem is hard

Diffie-Hellman Key Exchange: Security (DLog)

The adversary sees (or knows already) G, q, g

The adversary sees $h_A = g^x$ and $h_B = g^y$

The shared key is $k = g^{xy}$

If the Discrete-Logarithm assumption does not hold, then computing discrete logarithms is easy for all groups output by poly-time group generation algorithms

In particular, it is easy for our group G

Diffie-Hellman Key Exchange: Security (DLog)

The adversary sees (or knows already) G, q, g

The adversary sees $h_A = g^x$ and $h_B = g^y$

The shared key is $k = g^{xy}$

If the Discrete-Logarithm assumption does not hold, then computing discrete logarithms is easy for all groups output by poly-time group generation algorithms

In particular, it is easy for our group G

- The adversary knows $h_A = g^x$
- It can recover $x = \log_g h_A$ with non-negligible probability
- Once x is known, it can compute $h_B^x = g^{xy} = k$

Diffie-Hellman Key Exchange: Security (DLog)

The adversary sees (or knows already) G, q, g

The adversary sees $h_A = g^x$ and $h_B = g^y$

The shared key is $k = g^{xy}$

If the Discrete-Logarithm assumption does not hold, then computing discrete logarithms is easy for all groups output by poly-time group generation algorithms

In particular, it is easy for our group G

- The adversary knows $h_A = g^x$
- It can recover $x = \log_g h_A$ with non-negligible probability
- Once x is known, it can compute $h_B^x = g^{xy} = k$

The Discrete-Logarithm assumption is a **necessary condition**

Diffie-Hellman Key Exchange: Security (CDH)

The adversary sees (or knows already) G, q, g

The adversary sees $h_A = g^x$ and $h_B = g^y$

The shared key is $k = g^{xy}$

If the CDH assumption does not hold, then computing $\mathsf{DH}(\cdot, \cdot)$ is easy for all groups output by poly-time group generation algorithms

In particular, it is easy for our group G

Diffie-Hellman Key Exchange: Security (CDH)

The adversary sees (or knows already) G, q, g

The adversary sees $h_A = g^x$ and $h_B = g^y$

The shared key is $k = g^{xy}$

If the CDH assumption does not hold, then computing $\mathsf{DH}(\cdot, \cdot)$ is easy for all groups output by poly-time group generation algorithms

In particular, it is easy for our group G

- The adversary knows $h_A = g^x$ and $h_B = g^y$
- It can compute $\mathsf{DH}_g(h_A, h_B) = g^{\log_g h_A \cdot \log_g h_B} = g^{xy} = k$ with non negligible probability

Diffie-Hellman Key Exchange: Security (CDH)

The adversary sees (or knows already) G, q, g

The adversary sees $h_A = g^x$ and $h_B = g^y$

The shared key is $k = g^{xy}$

If the CDH assumption does not hold, then computing DH$(\cdot, \cdot)$ is easy for all groups output by poly-time group generation algorithms

In particular, it is easy for our group G

- The adversary knows $h_A = g^x$ and $h_B = g^y$
- It can compute DH$_g(h_A, h_B) = g^{\log_g h_A \cdot \log_g h_B} = g^{xy} = k$ with non negligible probability

The CDH assumption is a **necessary condition**

Diffie-Hellman Key Exchange: Security (DDH)

The adversary sees (or knows already) G, q, g

The adversary sees $h_A = g^x$ and $h_B = g^y$

The shared key is $k = g^{xy}$

If the DDH assumption does not hold then, given g, g^x, and g^y, it possible to distinguish $k = g^{ab}$ from a group element chosen uniformly at random (for all groups output by poly-time group generation algorithms).

In particular, this holds for our group G.

Diffie-Hellman Key Exchange: Security (DDH)

The adversary sees (or knows already) G, q, g

The adversary sees $h_A = g^x$ and $h_B = g^y$

The shared key is $k = g^{xy}$

If the DDH assumption does not hold then, given g, g^x, and g^y, it possible to distinguish $k = g^{ab}$ from a group element chosen uniformly at random (for all groups output by poly-time group generation algorithms).

In particular, this holds for our group G.

- The adversary knows $h_A = g^x$ and $h_B = g^y$
- It can tell k apart from a random group element with a non-negligible advantage over random guessing
- It can use this to win the $\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ experiment

Diffie-Hellman Key Exchange: Security (DDH)

The adversary sees (or knows already) G, q, g

The adversary sees $h_A = g^x$ and $h_B = g^y$

The shared key is $k = g^{xy}$

If the DDH assumption does not hold then, given g, g^x, and g^y, it possible to distinguish $k = g^{ab}$ from a group element chosen uniformly at random (for all groups output by poly-time group generation algorithms).

In particular, this holds for our group G.

- The adversary knows $h_A = g^x$ and $h_B = g^y$
- It can tell k apart from a random group element with a non-negligible advantage over random guessing
- It can use this to win the $\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ experiment

The DDH assumption is a **necessary condition**

Diffie-Hellman Key Exchange: Security (Roadmap)

We will argue that the DDH assumption is also a **sufficient condition**

We first need to address a technical subtlety

- Intuitively, the DDH assumption guarantees that it is hard to distinguish $k = g^{xy}$ from a **random group element**
- The $\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ experiment asks to distinguish k from a **random** n**-bit string**

Diffie-Hellman Key Exchange: Security (Roadmap)

We will argue that the DDH assumption is also a **sufficient condition**

We first need to address a technical subtlety

- Intuitively, the DDH assumption guarantees that it is hard to distinguish $k = g^{xy}$ from a **random group element**
- The $\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ experiment asks to distinguish k from a **random n-bit string**
- In general, a random n-bit string is not even a group element
- $\implies$ It can be easy to tell a random n-bit string apart from k by simply testing whether it is a group element

Diffie-Hellman Key Exchange: Security (Roadmap)

We will argue that the DDH assumption is also a **sufficient condition**

We first need to address a technical subtlety

- Intuitively, the DDH assumption guarantees that it is hard to distinguish $k = g^{xy}$ from a **random group element**
- The $\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ experiment asks to distinguish k from a **random n-bit string**
- In general, a random n-bit string is not even a group element
- $\implies$ It can be easy to tell a random n-bit string apart from k by simply testing whether it is a group element

Solution (roadmap):

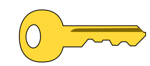
- We revise the $\mathsf{KE}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ experiment to take into account groups
- We prove security w.r.t. the revised experiment
- We argue that the shared key k can be turned into a (essentially random) n-bit string

Diffie-Hellman Key Exchange: Revised Experiment

Fix a group G, a key exchange protocol Π, and an attacker $\mathcal{A}$

We define the **key-exchange experiment** $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ as follows:

- The honest parties run Π using n as the security parameter.
- The interaction results in a transcript τ and in a shared key $k \in G$



- A random bit b is chosen u.a.r. from $\{0, 1\}$.
  - If $b = 0$ then $k' \leftarrow k$.
  - Otherwise (when $b = 1$) k' is chosen as a random element from G.
- $\mathcal{A}$ is given k' and the transcript τ

- $\mathcal{A}$ outputs a bit $b' \in \{0, 1\}$

The outcome of the experiment is defined to be 1 if $b' = b$ and 0 otherwise

Diffie-Hellman Key Exchange: Security Proof

Theorem: If the DDH problem is hard relative to G, then the Diffie–Hellman key-exchange protocol Π is secure in the presence of an eavesdropper with respect to the modified experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$

Proof:

We show that a probabilistic polynomial-time adversary $\mathcal{A}$ that wins the experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ with non-negligible advantage, can be used to design a probabilistic polynomial-time algorithm that solves the DDH problem with non-negligible gap.

- Suppose towards a contradiction that there exists $\mathcal{A}$ such that $\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.

Diffie-Hellman Key Exchange: Security Proof

Theorem: If the DDH problem is hard relative to G, then the Diffie–Hellman key-exchange protocol Π is secure in the presence of an eavesdropper with respect to the modified experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$

Proof:

We show that a probabilistic polynomial-time adversary $\mathcal{A}$ that wins the experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ with non-negligible advantage, can be used to design a probabilistic polynomial-time algorithm that solves the DDH problem with non-negligible gap.

- Suppose towards a contradiction that there exists $\mathcal{A}$ such that $\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.
- To solve the DDH problem... we use $\mathcal{A}$ itself!

Diffie-Hellman Key Exchange: Security Proof

Theorem: If the DDH problem is hard relative to G, then the Diffie–Hellman key-exchange protocol Π is secure in the presence of an eavesdropper with respect to the modified experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$

Proof:

We show that a probabilistic polynomial-time adversary $\mathcal{A}$ that wins the experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ with non-negligible advantage, can be used to design a probabilistic polynomial-time algorithm that solves the DDH problem with non-negligible gap.

- Suppose towards a contradiction that there exists $\mathcal{A}$ such that $\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.
- To solve the DDH problem... we use $\mathcal{A}$ itself!

$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1]$

Diffie-Hellman Key Exchange: Security Proof

Theorem: If the DDH problem is hard relative to G, then the Diffie–Hellman key-exchange protocol Π is secure in the presence of an eavesdropper with respect to the modified experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$

Proof:

We show that a probabilistic polynomial-time adversary $\mathcal{A}$ that wins the experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ with non-negligible advantage, can be used to design a probabilistic polynomial-time algorithm that solves the DDH problem with non-negligible gap.

- Suppose towards a contradiction that there exists $\mathcal{A}$ such that $\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.
- To solve the DDH problem... we use $\mathcal{A}$ itself!

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] = \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

Diffie-Hellman Key Exchange: Security Proof

Theorem: If the DDH problem is hard relative to G, then the Diffie–Hellman key-exchange protocol Π is secure in the presence of an eavesdropper with respect to the modified experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$

Proof:

We show that a probabilistic polynomial-time adversary $\mathcal{A}$ that wins the experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ with non-negligible advantage, can be used to design a probabilistic polynomial-time algorithm that solves the DDH problem with non-negligible gap.

- Suppose towards a contradiction that there exists $\mathcal{A}$ such that $\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.
- To solve the DDH problem... we use $\mathcal{A}$ itself!

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] = \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

Diffie-Hellman Key Exchange: Security Proof

Theorem: If the DDH problem is hard relative to G, then the Diffie–Hellman key-exchange protocol Π is secure in the presence of an eavesdropper with respect to the modified experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$

Proof:

We show that a probabilistic polynomial-time adversary $\mathcal{A}$ that wins the experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ with non-negligible advantage, can be used to design a probabilistic polynomial-time algorithm that solves the DDH problem with non-negligible gap.

- Suppose towards a contradiction that there exists $\mathcal{A}$ such that $\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.

- To solve the DDH problem... we use $\mathcal{A}$ itself!

$k' = g^z$ for some z chosen u.a.r. from $\{0, \ldots, q-1\}$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] = \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

Diffie-Hellman Key Exchange: Security Proof

Theorem: If the DDH problem is hard relative to G, then the Diffie–Hellman key-exchange protocol Π is secure in the presence of an eavesdropper with respect to the modified experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$

Proof:

We show that a probabilistic polynomial-time adversary $\mathcal{A}$ that wins the experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ with non-negligible advantage, can be used to design a probabilistic polynomial-time algorithm that solves the DDH problem with non-negligible gap.

- Suppose towards a contradiction that there exists $\mathcal{A}$ such that $\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.

- To solve the DDH problem... we use $\mathcal{A}$ itself!

$k' = g^z$ for some z chosen u.a.r. from $\{0, \ldots, q-1\}$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] = \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0]$$

Diffie-Hellman Key Exchange: Security Proof

Theorem: If the DDH problem is hard relative to G, then the Diffie–Hellman key-exchange protocol Π is secure in the presence of an eavesdropper with respect to the modified experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$

Proof:

We show that a probabilistic polynomial-time adversary $\mathcal{A}$ that wins the experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ with non-negligible advantage, can be used to design a probabilistic polynomial-time algorithm that solves the DDH problem with non-negligible gap.

- Suppose towards a contradiction that there exists $\mathcal{A}$ such that $\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.
- To solve the DDH problem... we use $\mathcal{A}$ itself!

$k' = g^z$ for some z chosen u.a.r. from $\{0, \ldots, q-1\}$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] = \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] = \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 0]$$

Diffie-Hellman Key Exchange: Security Proof

Theorem: If the DDH problem is hard relative to G, then the Diffie–Hellman key-exchange protocol Π is secure in the presence of an eavesdropper with respect to the modified experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$

Proof:

We show that a probabilistic polynomial-time adversary $\mathcal{A}$ that wins the experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ with non-negligible advantage, can be used to design a probabilistic polynomial-time algorithm that solves the DDH problem with non-negligible gap.

- Suppose towards a contradiction that there exists $\mathcal{A}$ such that $\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.
- To solve the DDH problem... we use $\mathcal{A}$ itself!

$k' = g^z$ for some z chosen u.a.r. from $\{0, \ldots, q-1\}$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] = \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] = \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 0]$$

$k' = k = g^{xy}$

Diffie-Hellman Key Exchange: Security Proof

Theorem: If the DDH problem is hard relative to G, then the Diffie–Hellman key-exchange protocol Π is secure in the presence of an eavesdropper with respect to the modified experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$

Proof:

We show that a probabilistic polynomial-time adversary $\mathcal{A}$ that wins the experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$ with non-negligible advantage, can be used to design a probabilistic polynomial-time algorithm that solves the DDH problem with non-negligible gap.

- Suppose towards a contradiction that there exists $\mathcal{A}$ such that $\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2} + \varepsilon(n)$ for non-negligible $\varepsilon(n)$.
- To solve the DDH problem... we use $\mathcal{A}$ itself!

$k' = g^z$ for some z chosen u.a.r. from $\{0, \ldots, q-1\}$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] = \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] = \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 0] \quad = 1 - \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1]$$

$k' = k = g^{xy}$

Diffie-Hellman Key Exchange: Security Proof (cont.)

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] = \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] = 1 - \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1]$$

Diffie-Hellman Key Exchange: Security Proof (cont.)

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] \;= \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] \;= 1 - \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1]$$

$$\tfrac{1}{2} + \varepsilon(n) = \Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \tfrac{1}{2}\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] + \tfrac{1}{2}\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1]$$

Diffie-Hellman Key Exchange: Security Proof (cont.)

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] \;\; = \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] \;\; = 1 - \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1]$$

$$\frac{1}{2} + \varepsilon(n) = \Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \frac{1}{2}\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] + \frac{1}{2}\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1]$$

$$= \frac{1}{2} - \frac{1}{2} \cdot \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] + \frac{1}{2} \cdot \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

Diffie-Hellman Key Exchange: Security Proof (cont.)

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] \;= \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] \;= 1 - \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1]$$

$$\begin{aligned}
\tfrac{1}{2} + \varepsilon(n) = \; & \Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \tfrac{1}{2}\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] + \tfrac{1}{2}\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] \\
= \; & \tfrac{1}{2} - \tfrac{1}{2} \cdot \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] + \tfrac{1}{2} \cdot \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1] \\
= \; & \tfrac{1}{2} + \tfrac{1}{2} \cdot (-\Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] + \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1])
\end{aligned}$$

Diffie-Hellman Key Exchange: Security Proof (cont.)

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] \;= \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] \;= 1 - \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1]$$

$$\begin{aligned}
\tfrac{1}{2} + \varepsilon(n) = \; & \Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \tfrac{1}{2}\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] + \tfrac{1}{2}\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] \\
& = \tfrac{1}{2} - \tfrac{1}{2} \cdot \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] + \tfrac{1}{2} \cdot \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1] \\
& = \tfrac{1}{2} + \tfrac{1}{2} \cdot (-\Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] + \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]) \\
& \leq \tfrac{1}{2} + \tfrac{1}{2} \cdot \Big| \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1] \Big|
\end{aligned}$$

Diffie-Hellman Key Exchange: Security Proof (cont.)

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] = \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] = 1 - \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1]$$

$$\begin{aligned}
\tfrac{1}{2} + \varepsilon(n) = \Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] &= \tfrac{1}{2}\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] + \tfrac{1}{2}\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] \\
&= \tfrac{1}{2} - \tfrac{1}{2} \cdot \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] + \tfrac{1}{2} \cdot \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1] \\
&= \tfrac{1}{2} + \tfrac{1}{2} \cdot (-\Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] + \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]) \\
&\leq \tfrac{1}{2} + \tfrac{1}{2} \cdot \Big| \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1] \Big|
\end{aligned}$$

$$\Big| \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1] \Big| \geq 2\varepsilon(n)$$

Diffie-Hellman Key Exchange: Security Proof (cont.)

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] \;\; = \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]$$

$$\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] \;\; = 1 - \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1]$$

$$\begin{aligned}
\tfrac{1}{2} + \varepsilon(n) = \; & \Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1] = \tfrac{1}{2}\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 0] + \tfrac{1}{2}\Pr[\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n) = 1 \mid b = 1] \\
& = \tfrac{1}{2} - \tfrac{1}{2} \cdot \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] + \tfrac{1}{2} \cdot \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1] \\
& = \tfrac{1}{2} + \tfrac{1}{2} \cdot (-\Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] + \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1]) \\
& \leq \tfrac{1}{2} + \tfrac{1}{2} \cdot \left| \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1] \right|
\end{aligned}$$

$$\left| \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^{xy}) = 1] - \Pr[\mathcal{A}(G, q, g, g^x, g^y, g^z) = 1] \right| \geq 2\varepsilon(n)$$

Not negligible

$\square$

Turning a random group element into a random binary string

We have shown the (conditional) security of the Diffie-Hellman protocol with respect to the modified experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$

The Diffie-Hellman key exchange returns a group element $k = g^{xy}$ that is indistinguishable from a random one to any polynomial-time adversary

What we actually need is a shared **key** k^*, i.e., a n-bit binary string indistinguishable from a random string (to any polynomial-time adversary)

How do we do that?

Turning a random group element into a random binary string

We have shown the (conditional) security of the Diffie-Hellman protocol with respect to the modified experiment $\widehat{\mathsf{KE}}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$

The Diffie-Hellman key exchange returns a group element $k = g^{xy}$ that is indistinguishable from a random one to any polynomial-time adversary

What we actually need is a shared **key** k^*, i.e., a n-bit binary string indistinguishable from a random string (to any polynomial-time adversary)

How do we do that?

Key derivation! Use a hash function H and set $k^* = H(k)$

This is secure if we model H as a random oracle

Man-in-the-Middle Attack

So far we have considered eavesdropping adversaries

What about active adversaries?

Man-in-the-Middle Attack

So far we have considered eavesdropping adversaries

What about active adversaries?

The Diffie-Hellman Key Exchange is vulnerable to the Man-in-the-Middle attack

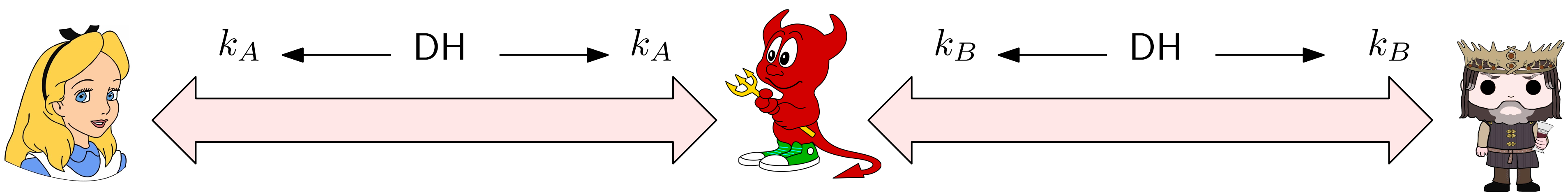
Man-in-the-Middle Attack

So far we have considered eavesdropping adversaries

What about active adversaries?

The Diffie-Hellman Key Exchange is vulnerable to the Man-in-the-Middle attack

- And adversary interacts with both Alice and Bob
- On Alice's side, the adversary pretends to be Bob and runs Bob's side of the Diffie-Hellman protocol
- On Bob's side, the adversary pretends to be Alice and runs Alice's side of the Diffie-Hellman protocol



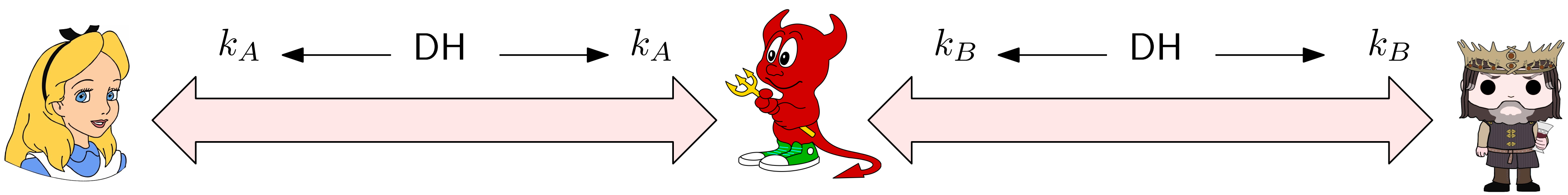
Man-in-the-Middle Attack

So far we have considered eavesdropping adversaries

What about active adversaries?

The Diffie-Hellman Key Exchange is vulnerable to the Man-in-the-Middle attack

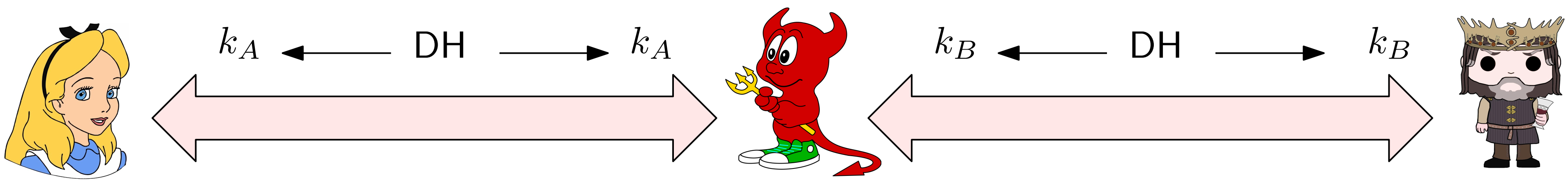
- And adversary interacts with both Alice and Bob
- On Alice's side, the adversary pretends to be Bob and runs Bob's side of the Diffie-Hellman protocol
- On Bob's side, the adversary pretends to be Alice and runs Alice's side of the Diffie-Hellman protocol



- At the end of the protocol Alice and Bob have two different keys k_A, k_B and both of these keys are known to the Adversary
- When a message is sent from A to B (or vice-versa) the adversary can decrypt it with k_A, read and possibily alter the plaintext, and re-encrypt it with k_B

Man-in-the-Middle Attack

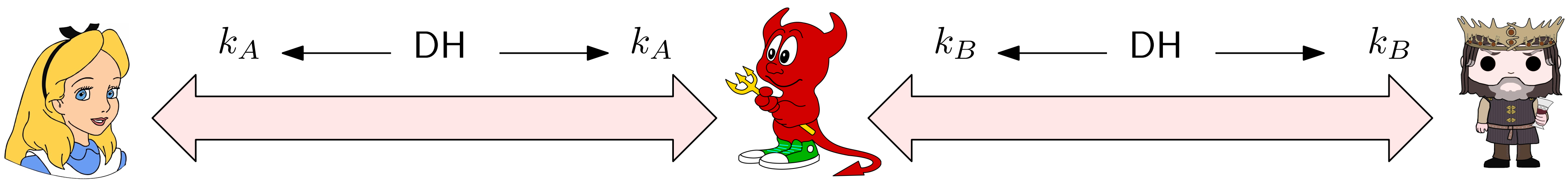
What can we do against this attack?



Man-in-the-Middle Attack

What can we do against this attack?

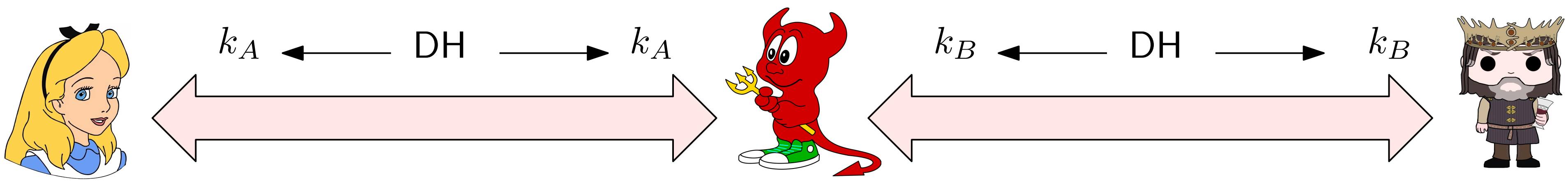
Man-in-the-Middle attacks can't be avoided if Alice and Bob don't know anything about each other and don't trust any 3rd-party



Man-in-the-Middle Attack

What can we do against this attack?

Man-in-the-Middle attacks can't be avoided if Alice and Bob don't know anything about each other and don't trust any 3rd-party



- In general we want authenticated key exchange protocols
- Can be achieved with public-key cryptography and digital signatures/certificates
- Modern key-exchange protocols (e.g., TLS) provide authentication

Private-Key Setting

Two (or more) parties who wish to communicate secretly need to share a uniform secret key k in advance

The same key can be used for both sending and receiving

- Each party can both send an receive
- If multiple parties share the same key, there is no way to distinguish them

The key must be kept secret!

- If an attacker gets to know k we lose all the security guarantees

Public-Key Setting

One party generates a pair of keys:

- A **public key** pk
- A **secret key** (or private key) sk

Public-Key Setting

One party generates a pair of keys:

- A **public key** pk
- A **secret key** (or private key) sk

The public key is. . . public

- No need to keep it secret
- In fact it is often widely disseminated
- Used by anybody that wishes to send messages to the party
- Security must hold even if an attacker knows pk

Public-Key Setting

One party generates a pair of keys:

- A **public key** pk
- A **secret key** (or private key) sk

The public key is. . . public

- No need to keep it secret
- In fact it is often widely disseminated
- Used by anybody that wishes to send messages to the party
- Security must hold even if an attacker knows pk

The secret key must be kept. . . secret

- It is only used by the party that generated it

Public-Key Setting

We assume that the involved parties are able to obtain (untampered) copies of the public keys

- The attacker is just an eavesdropper; or
- The attacker remains passive *during key distribution*

Public-Key Setting

We assume that the involved parties are able to obtain (untampered) copies of the public keys

- The attacker is just an eavesdropper; or
- The attacker remains passive *during key distribution*

Advantages:

- **Key distribution:** Keys can be distributed over a public channel

Public-Key Setting

We assume that the involved parties are able to obtain (untampered) copies of the public keys

- The attacker is just an eavesdropper; or
- The attacker remains passive *during key distribution*

Advantages:

- **Key distribution:** Keys can be distributed over a public channel
- **Key management:** Each user needs a single public/secret key pair.
  - In a system with N users, there are $\Theta(N)$ key pairs, instead of $\Theta(N^2)$ private-keys

Public-Key Setting

We assume that the involved parties are able to obtain (untampered) copies of the public keys

- The attacker is just an eavesdropper; or
- The attacker remains passive *during key distribution*

Advantages:

- **Key distribution:** Keys can be distributed over a public channel
- **Key management:** Each user needs a single public/secret key pair.
  - In a system with N users, there are $\Theta(N)$ key pairs, instead of $\Theta(N^2)$ private-keys
- **Open systems:** Two parties with no prior relationship can find each others' public keys*
  - The recipient does not even need to know who the sender is

*Requires a trusted third party

Public-Key Setting

Why study private key cryptography at all?

- If two parties wish to communicate, they can always generate two public/secret key pairs instead of a secret shared key

But...

Public-Key Setting

Why study private key cryptography at all?

- If two parties wish to communicate, they can always generate two public/secret key pairs instead of a secret shared key

But...

- Public-key schemes are slower

Public-Key Setting

Why study private key cryptography at all?

- If two parties wish to communicate, they can always generate two public/secret key pairs instead of a secret shared key

But...

- Public-key schemes are slower
- Public-key schemes require longer keys

Public-Key Setting

Why study private key cryptography at all?

- If two parties wish to communicate, they can always generate two public/secret key pairs instead of a secret shared key

But...

- Public-key schemes are slower
- Public-key schemes require longer keys
- Public-key schemes have larger ciphertext expansion

Public-Key Setting

Why study private key cryptography at all?

- If two parties wish to communicate, they can always generate two public/secret key pairs instead of a secret shared key

But...

- Public-key schemes are slower
- Public-key schemes require longer keys
- Public-key schemes have larger ciphertext expansion
- Public-key schemes require stronger assumptions

Public-Key Setting

Why study private key cryptography at all?

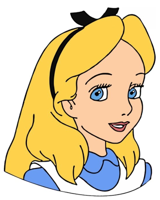
- If two parties wish to communicate, they can always generate two public/secret key pairs instead of a secret shared key

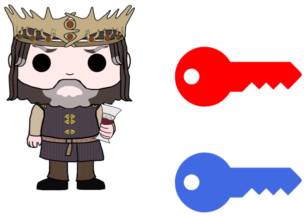
But...

- Public-key schemes are slower
- Public-key schemes require longer keys
- Public-key schemes have larger ciphertext expansion
- Public-key schemes require stronger assumptions

These issues can be mitigated

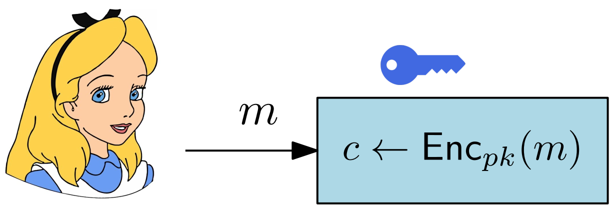
Public-Key Encryption

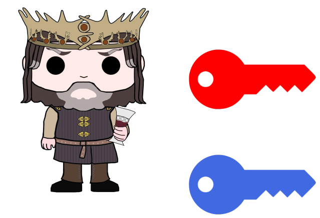




- Bob generates a public/secret key pair (pk, sk) 
- Bob shares pk with Alice 

Public-Key Encryption

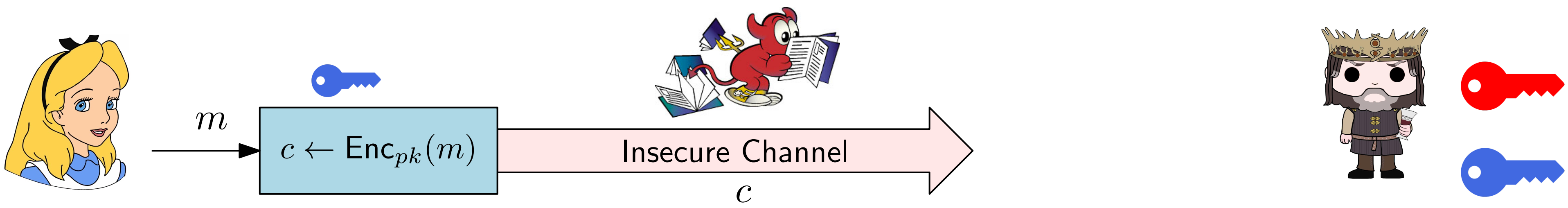




- Bob generates a public/secret key pair (pk, sk)
- Bob shares pk with Alice
- Alice encrypts her message m using Bob's public key pk

$$c \leftarrow \mathsf{Enc}_{pk}(m)$$

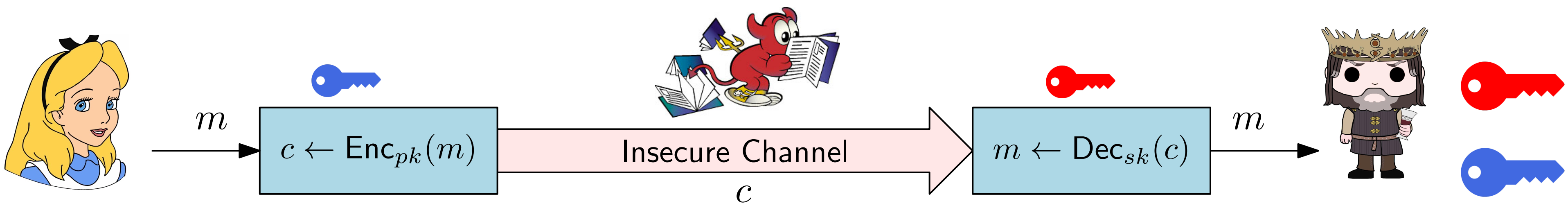
Public-Key Encryption



- Bob generates a public/secret key pair (pk, sk)
- Bob shares pk with Alice
- Alice encrypts her message m using Bob's public key pk

$$c \leftarrow \mathsf{Enc}_{pk}(m)$$

Public-Key Encryption



- Bob generates a public/secret key pair (pk, sk)
- Bob shares pk with Alice
- Alice encrypts her message m using Bob's public key pk

$$c \leftarrow \mathsf{Enc}_{pk}(m)$$

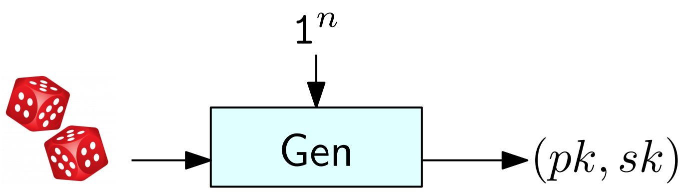
Bob **decrypts** the ciphertext c using his secret key sk

$$m \leftarrow \mathsf{Dec}_{sk}(c)$$

Public-key Encryption Schemes: Definition

A **public-key encryption scheme** consists of three algorithms:

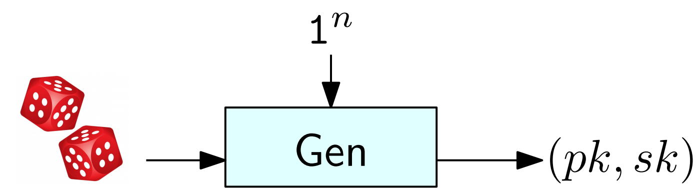
- Gen is a randomized algorithm that takes 1^n as input and outputs a pair of keys (pk, sk), each of length at least n. The public key defines a message space $\mathcal{M}_{pk}$



Public-key Encryption Schemes: Definition

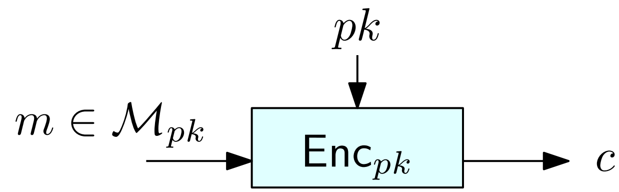
A **public-key encryption scheme** consists of three algorithms:

- Gen is a randomized algorithm that takes 1^n as input and outputs a pair of keys (pk, sk), each of length at least n. The public key defines a message space $\mathcal{M}_{pk}$



- Enc is a randomized algorithm that takes as input a public key pk and a *message* (or plaintext) $m \in \mathcal{M}_{pk}$ and outputs a *ciphertext* c obtained by encrypting m with key pk.

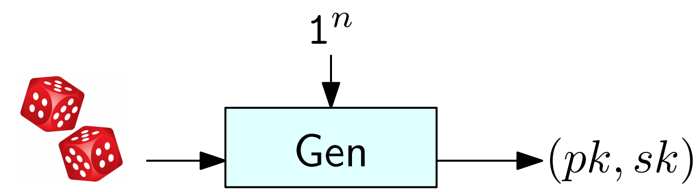
  $\mathsf{Enc}_{pk}(m)$ denotes an execution of Enc with inputs pk and m



Public-key Encryption Schemes: Definition

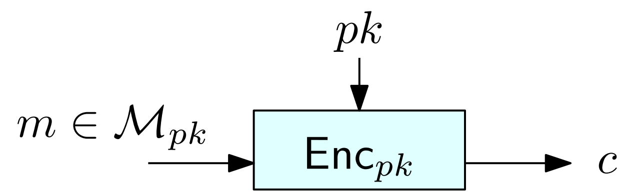
A **public-key encryption scheme** consists of three algorithms:

- Gen is a randomized algorithm that takes 1^n as input and outputs a pair of keys (pk, sk), each of length at least n. The public key defines a message space $\mathcal{M}_{pk}$



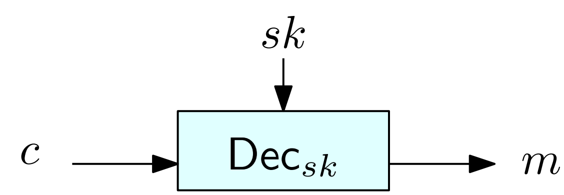
- Enc is a randomized algorithm that takes as input a public key pk and a *message* (or plaintext) $m \in \mathcal{M}_{pk}$ and outputs a *ciphertext* c obtained by encrypting m with key pk.

  $\mathsf{Enc}_{pk}(m)$ denotes an execution of Enc with inputs pk and m



- Dec is a deterministic algorithm that takes as input a secret key sk and a ciphertext c and outputs a message $m \in \mathcal{M}_{pk}$ or a special symbol $\perp$ denoting failure.

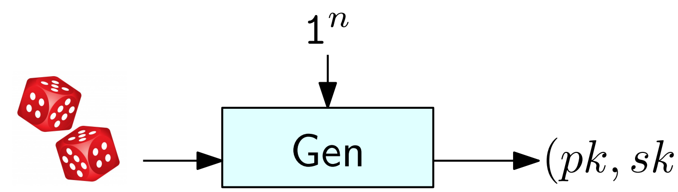
  $\mathsf{Dec}_{sk}(c)$ denotes an execution of Dec with inputs sk and c



Public-key Encryption Schemes: Definition

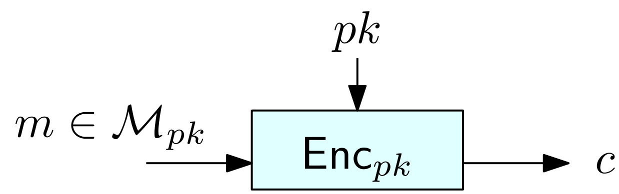
A **public-key encryption scheme** consists of three algorithms:

- Gen is a randomized algorithm that takes 1^n as input and outputs a pair of keys (pk, sk), each of length at least n. The public key defines a message space $\mathcal{M}_{pk}$



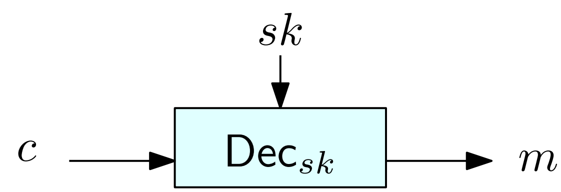
- Enc is a randomized algorithm that takes as input a public key pk and a *message* (or plaintext) $m \in \mathcal{M}_{pk}$ and outputs a *ciphertext* c obtained by encrypting m with key pk.

  $\text{Enc}_{pk}(m)$ denotes an execution of Enc with inputs pk and m



- Dec is a deterministic algorithm that takes as input a secret key sk and a ciphertext c and outputs a message $m \in \mathcal{M}_{pk}$ or a special symbol $\perp$ denoting failure.

  $\text{Dec}_{sk}(c)$ denotes an execution of Dec with inputs sk and c



Correctness: We must have $\text{Dec}_{sk}(\,\text{Enc}_{pk}(m)\,) = m$ for any $m \in \mathcal{M}_{pk}$, except for negligible probability (over the randomness of Gen and Enc).

Security Notion for Public-Key Encryption Schemes?

How do we formalize the notion of security for Public-Key encryption?

- We can hope to adapt the security definition that we used for Private-Key encryption schemes

Security Notion for Public-Key Encryption Schemes?

How do we formalize the notion of security for Public-Key encryption?

- We can hope to adapt the security definition that we used for Private-Key encryption schemes

- Let's start simple... security in the presence of an eavesdropper

Security Notion for Public-Key Encryption Schemes?

How do we formalize the notion of security for Public-Key encryption?

- We can hope to adapt the security definition that we used for Private-Key encryption schemes
- Let's start simple... security in the presence of an eavesdropper

Experiment $\mathsf{PrivK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$:

- A random key k is generated (by running Gen)
- $\mathcal{A}$ chooses two distinct messages $m_0, m_1 \in \mathcal{M}$
- A uniform random bit $b \in \{0, 1\}$ is generated
- The *challenge ciphertext* $c = \mathsf{Enc}_k(m_b)$, and given to $\mathcal{A}$
- $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ about b
- The *outcome of the experiment* is defined to be 1 if $b' = b$, and 0 otherwise

Security Notion for Public-Key Encryption Schemes?

How do we formalize the notion of security for Public-Key encryption?

- We can hope to adapt the security definition that we used for Private-Key encryption schemes
- Let's start simple... security in the presence of an eavesdropper

Experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$:

- A random key k is generated (by running Gen)
- $\mathcal{A}$ chooses two distinct messages $m_0, m_1 \in \mathcal{M}$
- A uniform random bit $b \in \{0,1\}$ is generated
- The *challenge ciphertext* $c = \mathsf{Enc}_k(m_b)$, and given to $\mathcal{A}$
- $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$ about b
- The *outcome of the experiment* is defined to be 1 if $b' = b$, and 0 otherwise

Security Notion for Public-Key Encryption Schemes?

How do we formalize the notion of security for Public-Key encryption?

- We can hope to adapt the security definition that we used for Private-Key encryption schemes
- Let's start simple... security in the presence of an eavesdropper

Experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$:

- A random key pair $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ is generated
- $\mathcal{A}$ chooses two distinct messages $m_0, m_1 \in \mathcal{M}$
- A uniform random bit $b \in \{0, 1\}$ is generated
- The *challenge ciphertext* $c = \mathsf{Enc}_k(m_b)$, and given to $\mathcal{A}$
- $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ about b
- The *outcome of the experiment* is defined to be 1 if $b' = b$, and 0 otherwise

Security Notion for Public-Key Encryption Schemes?

How do we formalize the notion of security for Public-Key encryption?

- We can hope to adapt the security definition that we used for Private-Key encryption schemes

- Let's start simple... security in the presence of an eavesdropper

Experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$:

- A random key pair $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ is generated
- The public key pk is sent to $\mathcal{A}$
- $\mathcal{A}$ chooses two distinct messages $m_0, m_1 \in \mathcal{M}$
- A uniform random bit $b \in \{0, 1\}$ is generated
- The *challenge ciphertext* $c = \mathsf{Enc}_k(m_b)$, and given to $\mathcal{A}$
- $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ about b
- The *outcome of the experiment* is defined to be 1 if $b' = b$, and 0 otherwise

Security Notion for Public-Key Encryption Schemes?

How do we formalize the notion of security for Public-Key encryption?

- We can hope to adapt the security definition that we used for Private-Key encryption schemes

- Let's start simple... security in the presence of an eavesdropper

Experiment $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}(n)$:

- A random key pair $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ is generated
- The public key pk is sent to $\mathcal{A}$
- $\mathcal{A}$ chooses two distinct messages $m_0, m_1 \in \mathcal{M}$
- A uniform random bit $b \in \{0,1\}$ is generated
- The *challenge ciphertext* $c = \mathsf{Enc}_{pk}(m_b)$, and given to $\mathcal{A}$
- $\mathcal{A}$ outputs a guess $b' \in \{0,1\}$ about b
- The *outcome of the experiment* is defined to be 1 if $b' = b$, and 0 otherwise

Some observations on the $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$ experiment

The adversary has access to the public key pk

- It can choose m_0 and m_1 as a function of pk
- It can encrypt any message with pk by itself

Some observations on the $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$ experiment

The adversary has access to the public key pk

- It can choose m_0 and m_1 as a function of pk
- It can encrypt any message with pk by itself
- No need for an encryption oracle!

Some observations on the $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$ experiment

The adversary has access to the public key pk

- It can choose m_0 and m_1 as a function of pk
- It can encrypt any message with pk by itself
- No need for an encryption oracle!
- This definition already allows the attacker to perform chosen-plaintext attacks

Some observations on the $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$ experiment

The adversary has access to the public key pk

- It can choose m_0 and m_1 as a function of pk
- It can encrypt any message with pk by itself
- No need for an encryption oracle!
- This definition already allows the attacker to perform chosen-plaintext attacks

Let's revise the name... $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi} \longrightarrow \mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}$

Some observations on the $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi}$ experiment

The adversary has access to the public key pk

- It can choose m_0 and m_1 as a function of pk
- It can encrypt any message with pk by itself
- No need for an encryption oracle!
- This definition already allows the attacker to perform chosen-plaintext attacks

Let's revise the name... $\mathsf{PubK}^{\mathsf{eav}}_{\mathcal{A},\Pi} \longrightarrow \mathsf{PubK}^{\mathsf{cpa}}_{\mathcal{A},\Pi}$

Definition: *A public key encryption scheme* Π *is* **CPA-secure** *if, for every probabilistic polynomial-time adversary* $\mathcal{A}$, *there is a negligible function* ε *such that:*

$$\Pr[\mathit{PubK}^{\mathit{cpa}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \varepsilon(n)$$

Some consequences

Any public-key encryption scheme that is CPA-secure is also CPA-secure for multiple encryptions

(just like the private-key setting)

No deterministic public-key encryption scheme can be CPA-secure

(just like the private-key setting)

Some consequences

There is no such thing as a perfectly secret public key encryption scheme

Some consequences

There is no such thing as a perfectly secret public key encryption scheme

We show that an adversary can always decrypt a ciphertext $c \leftarrow \mathsf{Enc}_{pk}(m)$ using a brute-force attack.

(obviously, this adversary won't run in polynomial-time)

Some consequences

There is no such thing as a perfectly secret public key encryption scheme

We show that an adversary can always decrypt a ciphertext $c \leftarrow \mathsf{Enc}_{pk}(m)$ using a brute-force attack.

(obviously, this adversary won't run in polynomial-time)

Assume, for simplicity, that the decryption algorithm never fails.

Let R be an upper bound to the number of random bits used by the encryption algorithm

Some consequences

There is no such thing as a perfectly secret public key encryption scheme

We show that an adversary can always decrypt a ciphertext $c \leftarrow \mathsf{Enc}_{pk}(m)$ using a brute-force attack.

(obviously, this adversary won't run in polynomial-time)

Assume, for simplicity, that the decryption algorithm never fails.

Let R be an upper bound to the number of random bits used by the encryption algorithm

- The adversary enumerates all possible messages $m' \in \mathcal{M}_{pk}$:
  - For each m', the adversary enumerates all R-bit binary strings r
    - The adversary runs $\mathsf{Enc}_{pk}(m')$ using r as the random bits to obtain a ciphertext c'
    - If $c = c'$: Claim that the plaintext m is exactly m'. Stop.

Some consequences

There is no such thing as a perfectly secret public key encryption scheme

We show that an adversary can always decrypt a ciphertext $c \leftarrow \text{Enc}_{pk}(m)$ using a brute-force attack.

(obviously, this adversary won't run in polynomial-time)

Assume, for simplicity, that the decryption algorithm never fails.

Let R be an upper bound to the number of random bits used by the encryption algorithm

- The adversary enumerates all possible messages $m' \in \mathcal{M}_{pk}$:
  - For each m', the adversary enumerates all R-bit binary strings r
    - The adversary runs $\text{Enc}_{pk}(m')$ using r as the random bits to obtain a ciphertext c'
    - If $c = c'$: Claim that the plaintext m is exactly m'. Stop.

Notice that:

- This algorithm must eventually stop (when $m' = m$ and r are the random bits used by $c \leftarrow \text{Enc}_{pk}(m)$)

Some consequences

There is no such thing as a perfectly secret public key encryption scheme

We show that an adversary can always decrypt a ciphertext $c \leftarrow \mathsf{Enc}_{pk}(m)$ using a brute-force attack.

(obviously, this adversary won't run in polynomial-time)

Assume, for simplicity, that the decryption algorithm never fails.

Let R be an upper bound to the number of random bits used by the encryption algorithm

- The adversary enumerates all possible messages $m' \in \mathcal{M}_{pk}$:
  - For each m', the adversary enumerates all R-bit binary strings r
    - The adversary runs $\mathsf{Enc}_{pk}(m')$ using r as the random bits to obtain a ciphertext c'
    - If $c = c'$: Claim that the plaintext m is exactly m'. Stop.

Notice that:

- This algorithm must eventually stop (when $m' = m$ and r are the random bits used by $c \leftarrow \mathsf{Enc}_{pk}(m)$)
- It cannot return $m' \neq m$ since this would imply $\mathsf{Enc}_{pk}(m') = c$ (for some randomness r) and hence $m = \mathsf{Dec}_{sk}(c) = \mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m'))$

Some consequences

There is no such thing as a perfectly secret public key encryption scheme

We show that an adversary can always decrypt a ciphertext $c \leftarrow \mathsf{Enc}_{pk}(m)$ using a brute-force attack.

(obviously, this adversary won't run in polynomial-time)

Assume, for simplicity, that the decryption algorithm never fails.

Let R be an upper bound to the number of random bits used by the encryption algorithm

- The adversary enumerates all possible messages $m' \in \mathcal{M}_{pk}$:
  - For each m', the adversary enumerates all R-bit binary strings r
    - The adversary runs $\mathsf{Enc}_{pk}(m')$ using r as the random bits to obtain a ciphertext c'
    - If $c = c'$: Claim that the plaintext m is exactly m'. Stop.

Notice that:

- This algorithm must eventually stop (when $m' = m$ and r are the random bits used by $c \leftarrow \mathsf{Enc}_{pk}(m)$)
- It cannot return $m' \neq m$ since this would imply $\mathsf{Enc}_{pk}(m') = c$ (for some randomness r) and hence $m = \mathsf{Dec}_{sk}(c) = \mathsf{Dec}_{sk}(\mathsf{Enc}_{pk}(m'))$

This contradicts correctness.

Security Against Chosen-Ciphertext Attacks

Chosen-Ciphertext attacks are also a concern in the public-key setting

In fact, they can even be easier to execute

Think of the following scenario:

- The adversary intercepts the encrypted body of an email from Alice to Bob
- The adversary sends the encrypted body to Bob from his address
- Bob's reply might reveal information about the plaintext
- If the adversary is lucky, the reply email will contain a full decryption of the original message!

Security Against Chosen-Ciphertext Attacks

Chosen-Ciphertext attacks are also a concern in the public-key setting

In fact, they can even be easier to execute

Think of the following scenario:

- The adversary intercepts the encrypted body of an email from Alice to Bob
- The adversary sends the encrypted body to Bob from his address
- Bob's reply might reveal information about the plaintext
- If the adversary is lucky, the reply email will contain a full decryption of the original message!

Bob is acting as a decryption oracle

Security Against Chosen-Ciphertext Attacks

Chosen-Ciphertext attacks are also a concern in the public-key setting

In fact, they can even be easier to execute

Think of the following scenario:

- The adversary intercepts the encrypted body of an email from Alice to Bob
- The adversary sends the encrypted body to Bob from his address
- Bob's reply might reveal information about the plaintext
- If the adversary is lucky, the reply email will contain a full decryption of the original message!

Bob is acting as a decryption oracle

It is possible to define CCA-security also for public-key encryption schemes

- In the private-key setting we have seen a stronger notion of security: **authenticated encryption**
- The definition of authenticated encryption does not immediately extend to the public-key setting (recall that anybody with the public key can encrypt messages)
- We will see another mechanism to guarantee authentication (digital signature schemes)

Definition of CCA-Security

Experiment $\mathsf{PubK}^{\mathsf{cca}}_{\mathcal{A},\Pi}(n)$:

- A random key pair $(pk, sk) \leftarrow \mathsf{Gen}(1^n)$ is generated and pk is sent to $\mathcal{A}$
- $\mathcal{A}$ interacts with a decryption oracle
- $\mathcal{A}$ chooses two distinct messages $m_0, m_1 \in \mathcal{M}$
- A uniform random bit $b \in \{0, 1\}$ is generated
- The *challenge ciphertext* c is computed by $\mathsf{Enc}_{pk}(m_b)$, and given to $\mathcal{A}$
- $\mathcal{A}$ interacts with a decryption oracle but cannot request a decryption of c
- $\mathcal{A}$ outputs a guess $b' \in \{0, 1\}$ about b
- The *outcome of the experiment* is defined to be 1 if $b' = b$, and 0 otherwise

Definition: *A public key encryption scheme Π is* **CCA-secure** *if, for every probabilistic polynomial-time adversary $\mathcal{A}$, there is a negligible function ε such that:*

$$\Pr[\mathit{PubK}^{\mathit{cca}}_{\mathcal{A},\Pi}(n) = 1] \leq \frac{1}{2} + \varepsilon(n)$$

Public-Key Setting vs. Private-Key Setting

	Private-Key Setting	Public-Key Setting
Secrecy	Private-Key Encryption Schemes	
Integrity	Message Authentication Codes	

Public-Key Setting vs. Private-Key Setting

	Private-Key Setting	Public-Key Setting
Secrecy	Private-Key Encryption Schemes	Public-Key Encryption Schemes
Integrity	Message Authentication Codes	

Public-Key Setting vs. Private-Key Setting

	Private-Key Setting	Public-Key Setting
Secrecy	Private-Key Encryption Schemes	Public-Key Encryption Schemes
Integrity	Message Authentication Codes	Digital Signature Schemes