Encryption II

601.442/642 Modern Cryptography

24th February 2026

Logistics

Logistics

- HW3 grades released

Logistics

- HW3 grades released
- Midterm grades out by Wednesday

Logistics

- HW3 grades released
- Midterm grades out by Wednesday
    - Ignore any intermediate grades released (sorry!)

Logistics

- HW3 grades released
- Midterm grades out by Wednesday
  - Ignore any intermediate grades released (sorry!)
  - One problem was way too hard, we will be giving everybody full credit for whatever their lowest grade on the 5 answered problems would be

Logistics

- HW3 grades released
- Midterm grades out by Wednesday
  - Ignore any intermediate grades released (sorry!)
  - One problem was way too hard, we will be giving everybody full credit for whatever their lowest grade on the 5 answered problems would be
  - i.e. you get 4 graded problems + 20 points.

Logistics

- HW3 grades released
- Midterm grades out by Wednesday
  - Ignore any intermediate grades released (sorry!)
  - One problem was way too hard, we will be giving everybody full credit for whatever their lowest grade on the 5 answered problems would be
  - i.e. you get 4 graded problems + 20 points.
- HW4 due Thursday

Logistics

- HW3 grades released
- Midterm grades out by Wednesday
  - Ignore any intermediate grades released (sorry!)
  - One problem was way too hard, we will be giving everybody full credit for whatever their lowest grade on the 5 answered problems would be
  - i.e. you get 4 graded problems + 20 points.
- HW4 due Thursday
- I am traveling next week, will not be holding office hours. Send me an email to schedule a specific time!

Recap: Public Key Encryption (PKE)

Public Key Encryption Scheme Syntax

A *public key encryption scheme* consists of three (possibly probabilistic) algorithms:

- KeyGen$(1^\lambda) \to (sk, pk)$ outputs a secret key $sk \in \mathcal{K}_s$ and a public key $pk \in \mathcal{K}_p$
- Enc$(pk, m) \to$ ct takes a public key pk and a message $m \in \mathcal{M}$ and outputs a ciphertext ct $\in \mathcal{C}$
- Dec$(sk,$ ct$) \to m$ takes a secret key sk and a ciphertext ct and outputs a message m

Recap: PKE Security

IND-CPA Security

A public key encryption scheme (KeyGen, Enc, Dec) satisfies *indistinguishability under chosen-plaintext attack* (IND-CPA) if for all NUPPT $\mathcal{A}$, there exists a negligible function $\nu(\cdot)$ such that $\forall \lambda \in \mathbb{N}$

$$\Pr[\mathcal{A} \text{ wins GuessGame}] \leq \frac{1}{2} + \nu(\lambda)$$

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

$(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)$

$\text{ct}_b \leftarrow \text{Enc}(pk, m_b)$

pk →

← (m_0, m_1)

ct_b →

← b'

Wins if $b' = b$

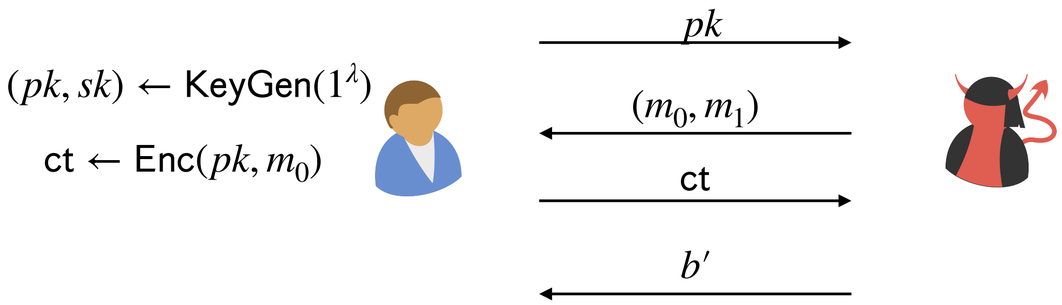
Recap: PKE Security

IND-CPA Security

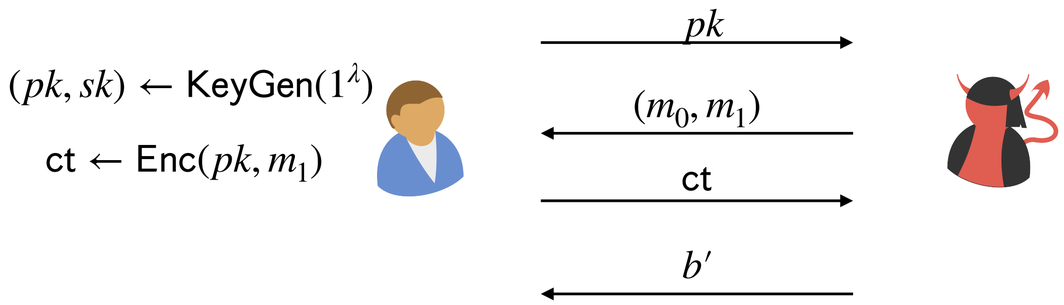
A public key encryption scheme (KeyGen, Enc, Dec) satisfies *indistinguishability under chosen-plaintext attack* (IND-CPA) if for all NUPPT $\mathscr{A}$, there exists a negligible function $\nu(\cdot)$ such that $\forall \lambda \in \mathbb{N}$

$$\left|\Pr[\mathscr{A} \text{ outputs } 1 \text{ in Game}_0] - \Pr[\mathscr{A} \text{ outputs } 1 \text{ in Game}_1]\right| \leq \nu(\lambda)$$

Game$_0$



Game$_1$



Recap: Cyclic Groups

Recap: Cyclic Groups

- A group is a set of elements and an operation: $(G, *)$

Recap: Cyclic Groups

- A group is a set of elements and an operation: $(G, *)$
- Notation: for $g \in G$, we have that $g^3 = g * g * g$

Recap: Cyclic Groups

- A group is a set of elements and an operation: $(G, *)$
- Notation: for $g \in G$, we have that $g^3 = g * g * g$
- Notation: let n be the *order* of the group, i.e. $n = |G|$

Recap: Cyclic Groups

- A group is a set of elements and an operation: $(G, *)$
- Notation: for $g \in G$, we have that $g^3 = g * g * g$
- Notation: let n be the *order* of the group, i.e. $n = |G|$
- A group $(G, *)$ is a *cyclic* group if there it is generated by a single element

Recap: Cyclic Groups

- A group is a set of elements and an operation: $(G, *)$
- Notation: for $g \in G$, we have that $g^3 = g * g * g$
- Notation: let n be the *order* of the group, i.e. $n = |G|$
- A group $(G, *)$ is a *cyclic* group if there it is generated by a single element
  - $G = \{1 = e = g^0, g^1, g^2, \ldots, g^{n-1}\}$

Recap: Cyclic Groups

- A group is a set of elements and an operation: $(G, *)$
- Notation: for $g \in G$, we have that $g^3 = g * g * g$
- Notation: let n be the *order* of the group, i.e. $n = |G|$
- A group $(G, *)$ is a *cyclic* group if there it is generated by a single element
  - $G = \{1 = e = g^0, g^1, g^2, \ldots, g^{n-1}\}$
  - g is a *generator* of G

Recap: Cyclic Groups

- A group is a set of elements and an operation: $(G, *)$
- Notation: for $g \in G$, we have that $g^3 = g * g * g$
- Notation: let n be the *order* of the group, i.e. $n = |G|$
- A group $(G, *)$ is a *cyclic* group if there it is generated by a single element
  - $G = \{1 = e = g^0, g^1, g^2, \ldots, g^{n-1}\}$
  - g is a *generator* of G
- We write this as $G = \langle g \rangle$

Recap: Cyclic Groups

- A group is a set of elements and an operation: $(G, *)$
- Notation: for $g \in G$, we have that $g^3 = g * g * g$
- Notation: let n be the *order* of the group, i.e. $n = |G|$
- A group $(G, *)$ is a *cyclic* group if there it is generated by a single element
  - $G = \{1 = e = g^0, g^1, g^2, \ldots, g^{n-1}\}$
  - g is a *generator* of G
- We write this as $G = \langle g \rangle$
- Our assumption: given g^a for some randomly sampled a, it should be hard to find a

Recap: Discrete Logarithm Assumption

Discrete Logarithm Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathcal{A} \text{ wins DLGame}\right] \leq \nu(\lambda)$$

$a \xleftarrow{\$} \{0, \ldots, p-1\}$

(G, p, g, g^a)

a'

Wins if $a' = a$

Recap: Discrete Logarithm Assumption

Discrete Logarithm Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathcal{A} \text{ wins DLGame}\right] \leq \nu(\lambda)$$

$a \xleftarrow{\$} \{0, \ldots, p-1\}$

(G, p, g, g^a)

a'

Wins if $a' = a$

Note: This is a *search problem*. $\mathcal{A}$ needs to actually *find* the discrete logarithm

Computational Diffie-Hellman Assumption

Computational Diffie-Hellman Assumption

- We haven’t figure out a way to make a public-key encryption scheme out of *just* the discrete log assumption.

Computational Diffie-Hellman Assumption

- We haven’t figure out a way to make a public-key encryption scheme out of *just* the discrete log assumption.
- Instead, we need to make a *stronger* assumption about what is hard for an adversary to do.

Computational Diffie-Hellman Assumption

- We haven't figure out a way to make a public-key encryption scheme out of *just* the discrete log assumption.
- Instead, we need to make a *stronger* assumption about what is hard for an adversary to do.
- Let $(G, *)$ be a cyclic group with generator g and order a safe prime p.

Computational Diffie-Hellman Assumption

- We haven't figure out a way to make a public-key encryption scheme out of *just* the discrete log assumption.
- Instead, we need to make a *stronger* assumption about what is hard for an adversary to do.
- Let $(G, *)$ be a cyclic group with generator g and order a safe prime p.
- Given (g^x, g^y) for randomly sampled x and y, it should be hard to find g^{xy}

Computational Diffie-Hellman Assumption

- We haven't figure out a way to make a public-key encryption scheme out of *just* the discrete log assumption.
- Instead, we need to make a *stronger* assumption about what is hard for an adversary to do.
- Let $(G, *)$ be a cyclic group with generator g and order a safe prime p.
- Given (g^x, g^y) for randomly sampled x and y, it should be hard to find g^{xy}
- Remember: $g^x * g^y = g^{x+y}$, so no trivial attack!

Computational Diffie-Hellman Assumption

Computational Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathcal{A} \text{ wins CDHGame}\right] \leq \nu(\lambda)$$

Computational Diffie-Hellman Assumption

Computational Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathcal{A} \text{ wins CDHGame}\right] \leq \nu(\lambda)$$

Computational Diffie-Hellman Assumption

Computational Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathscr{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathscr{A} \text{ wins CDHGame}\right] \leq \nu(\lambda)$$

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

Computational Diffie-Hellman Assumption

Computational Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathscr{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathscr{A} \text{ wins CDHGame}\right] \leq \nu(\lambda)$$

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$$(G, p, g, g^x, g^y) \longrightarrow$$

Computational Diffie-Hellman Assumption

Computational Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathscr{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathscr{A} \text{ wins CDHGame}\right] \leq \nu(\lambda)$$

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

(G, p, g, g^x, g^y)

z

Computational Diffie-Hellman Assumption

Computational Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathcal{A} \text{ wins CDHGame}\right] \leq \nu(\lambda)$$

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

(G, p, g, g^x, g^y)

z

Wins if $z = g^{xy}$

Computational Diffie-Hellman Assumption

Computational Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathcal{A} \text{ wins CDHGame}\right] \leq \nu(\lambda)$$

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

(G, p, g, g^x, g^y)

z

Wins if $z = g^{xy}$

Note: This is another *search problem*

Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman Assumption

- As it turns out, CDH is *still* not enough (on its own) to get us (efficient) public-key encryption.

Decisional Diffie-Hellman Assumption

- As it turns out, CDH is *still* not enough (on its own) to get us (efficient) public-key encryption.
- Remember: all previous constructions of encryption make the ciphertext look *totally random* via an xor with a random string.

Decisional Diffie-Hellman Assumption

- As it turns out, CDH is *still* not enough (on its own) to get us (efficient) public-key encryption.
- Remember: all previous constructions of encryption make the ciphertext look *totally random* via an xor with a random string.
- So, we need an assumption that gets us from *hard to find* (CDH) to *totally random*.

Decisional Diffie-Hellman Assumption

- As it turns out, CDH is *still* not enough (on its own) to get us (efficient) public-key encryption.
- Remember: all previous constructions of encryption make the ciphertext look *totally random* via an xor with a random string.
- So, we need an assumption that gets us from *hard to find* (CDH) to *totally random*.
- Let $(G, *)$ be a cyclic group with generator g and order a safe prime p.

Decisional Diffie-Hellman Assumption

- As it turns out, CDH is *still* not enough (on its own) to get us (efficient) public-key encryption.
- Remember: all previous constructions of encryption make the ciphertext look *totally random* via an xor with a random string.
- So, we need an assumption that gets us from *hard to find* (CDH) to *totally random*.
- Let $(G, *)$ be a cyclic group with generator g and order a safe prime p.
- Given (g^x, g^y) for randomly sampled x and y, g^{xy} should look *totally random*, i.e. be indistinguishable from g^r where r is randomly sampled.

Decisional Diffie-Hellman Assumption

- As it turns out, CDH is *still* not enough (on its own) to get us (efficient) public-key encryption.
- Remember: all previous constructions of encryption make the ciphertext look *totally random* via an xor with a random string.
- So, we need an assumption that gets us from *hard to find* (CDH) to *totally random*.
- Let $(G, *)$ be a cyclic group with generator g and order a safe prime p.
- Given (g^x, g^y) for randomly sampled x and y, g^{xy} should look *totally random*, i.e. be indistinguishable from g^r where r is randomly sampled.
- How do we model indistinguishability? A bit-guessing game!

Decisional Diffie-Hellman Assumption

- As it turns out, CDH is *still* not enough (on its own) to get us (efficient) public-key encryption.
- Remember: all previous constructions of encryption make the ciphertext look *totally random* via an xor with a random string.
- So, we need an assumption that gets us from *hard to find* (CDH) to *totally random*.
- Let $(G, *)$ be a cyclic group with generator g and order a safe prime p.
- Given (g^x, g^y) for randomly sampled x and y, g^{xy} should look *totally random*, i.e. be indistinguishable from g^r where *r* is randomly sampled.
- How do we model indistinguishability? A bit-guessing game!
- Challenger samples a bit b, samples (x, y, r), then either sends the adversary g^{xy} when $b = 0$, or g^r when $b = 1$. Adversary has to guess b.

Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathcal{A} \text{ wins DDHGame}\right] \leq \nu(\lambda)$$

Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathcal{A} \text{ wins DDHGame}\right] \leq \nu(\lambda)$$

Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathscr{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathscr{A} \text{ wins DDHGame}\right] \leq \nu(\lambda)$$

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

Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathscr{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathscr{A} \text{ wins DDHGame}\right] \leq \nu(\lambda)$$

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

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathcal{A} \text{ wins DDHGame}\right] \leq \nu(\lambda)$$

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

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathcal{A} \text{ wins DDHGame}\right] \leq \nu(\lambda)$$

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

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$r_0 := xy$

Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathscr{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathscr{A} \text{ wins DDHGame}\right] \leq \nu(\lambda)$$

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

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$r_0 := xy$

$r_1 \xleftarrow{\$} \{0,\ldots,p-1\}$

Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathcal{A} \text{ wins DDHGame}\right] \leq \nu(\lambda)$$

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

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$r_0 := xy$

$r_1 \xleftarrow{\$} \{0,\ldots,p-1\}$

$$(G, p, g, g^x, g^y, g^{r_b}) \longrightarrow$$

Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathscr{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathscr{A} \text{ wins DDHGame}\right] \leq \nu(\lambda)$$

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

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$r_0 := xy$

$r_1 \xleftarrow{\$} \{0,\ldots,p-1\}$

$(G, p, g, g^x, g^y, g^{r_b})$ →

← b'

Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathscr{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathscr{A} \text{ wins DDHGame}\right] \leq \nu(\lambda)$$

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

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$r_0 := xy$

$r_1 \xleftarrow{\$} \{0,\ldots,p-1\}$

$(G, p, g, g^x, g^y, g^{r_b})$ →

← b'

Wins if $b' = b$

Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathcal{A} \text{ wins DDHGame}\right] \leq \nu(\lambda)$$

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

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$r_0 := xy$

$r_1 \xleftarrow{\$} \{0,\ldots,p-1\}$

$(G, p, g, g^x, g^y, g^{r_b})$ →

← b'

Wins if $b' = b$

Note: This is a *decision problem*

DDH ⇒ CDH ⇒ DL

Decisional Diffie-Hellman Assumption

Decisional Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathcal{A}$, there exists a negligible function ν such that

$$\Pr\left[\mathcal{A} \text{ wins DDHGame}\right] \leq \nu(\lambda)$$

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

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$r_0 := xy$

$r_1 \xleftarrow{\$} \{0,\ldots,p-1\}$

$(G, p, g, g^x, g^y, g^{r_b})$ →

← b'

Wins if $b' = b$

Note: This is a *decision problem*

Decisional Diffie-Hellman Assumption

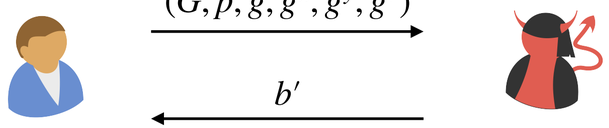
Decisional Diffie-Hellman Assumption

Let $(G, *)$ be a cyclic group of order p (where p is a safe prime) with generator g, then for every NUPPT adversary $\mathscr{A}$, there exists a negligible function ν such that

$$\left|\Pr[\mathscr{A} \text{ outputs } 1 \text{ in } \text{Game}_0] - \Pr[\mathscr{A} \text{ outputs } 1 \text{ in } \text{Game}_1]\right| \leq \nu(\lambda)$$

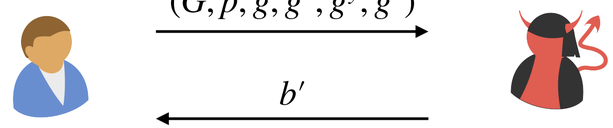
Game_0

$b \xleftarrow{\$} \{0,1\}$
$x \xleftarrow{\$} \{0,\ldots,p-1\}$
$y \xleftarrow{\$} \{0,\ldots,p-1\}$
$r := xy$



Game_1

$b \xleftarrow{\$} \{0,1\}$
$x \xleftarrow{\$} \{0,\ldots,p-1\}$
$y \xleftarrow{\$} \{0,\ldots,p-1\}$
$r \xleftarrow{\$} \{0,\ldots,p-1\}$



Key Exchange

Key Exchange

- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

Key Exchange

- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

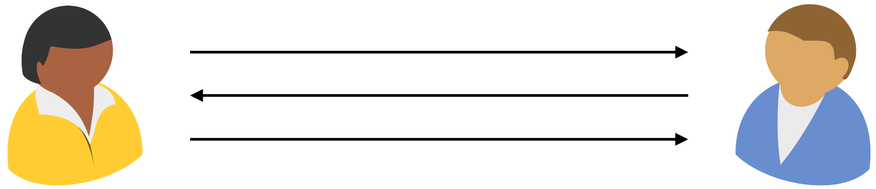
- Security: given a *view* of a protocol (i.e. all the public messages), the agreed key should be indistinguishable from random.

Key Exchange

- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

- Security: given a *view* of a protocol (i.e. all the public messages), the agreed key should be indistinguishable from random.

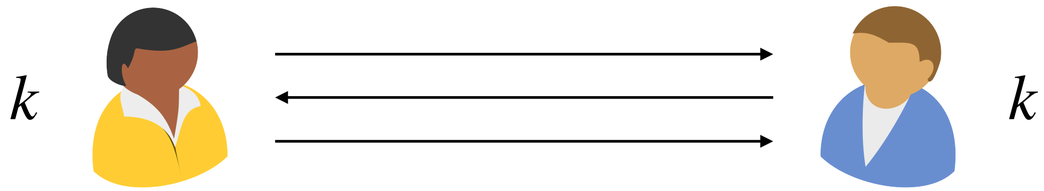
Key Exchange



- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

- Security: given a *view* of a protocol (i.e. all the public messages), the agreed key should be indistinguishable from random.

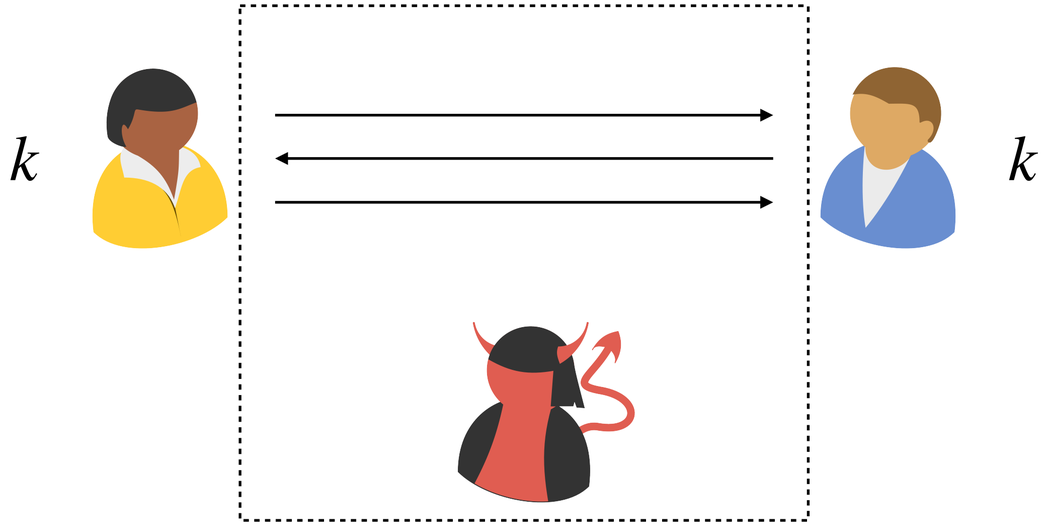
Key Exchange



- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

- Security: given a *view* of a protocol (i.e. all the public messages), the agreed key should be indistinguishable from random.

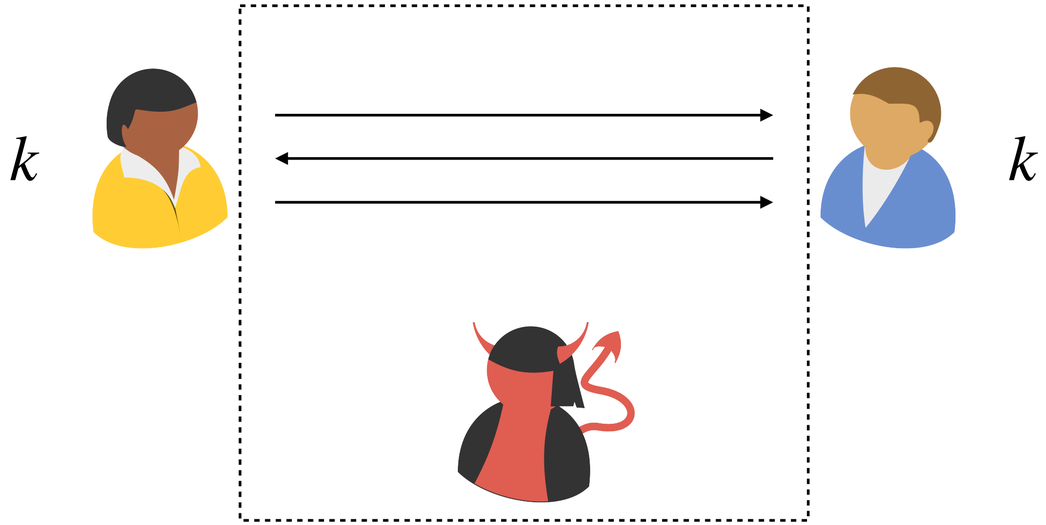
Key Exchange



- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

- Security: given a *view* of a protocol (i.e. all the public messages), the agreed key should be indistinguishable from random.

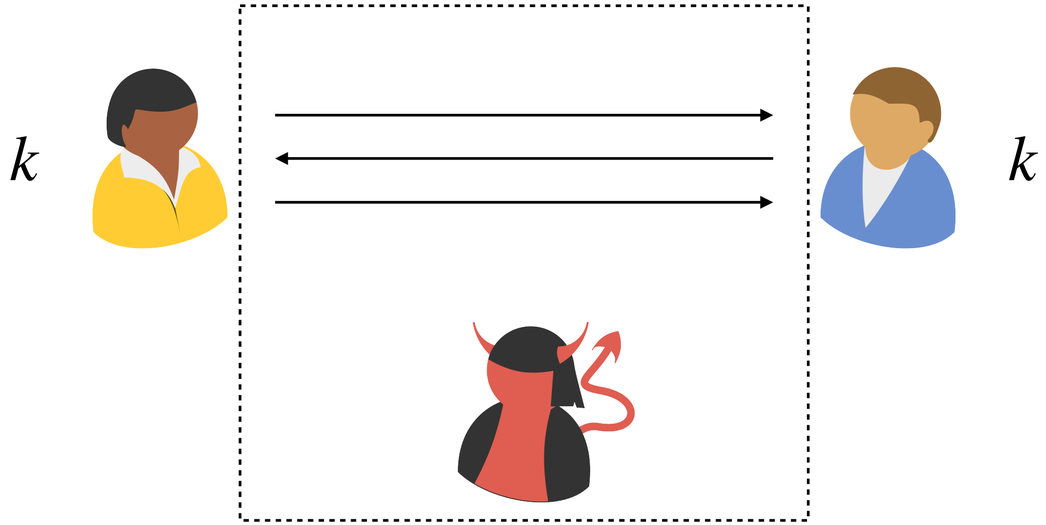
Key Exchange



- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

- Security: given a *view* of a protocol (i.e. all the public messages), the agreed key should be indistinguishable from random.

Key Exchange

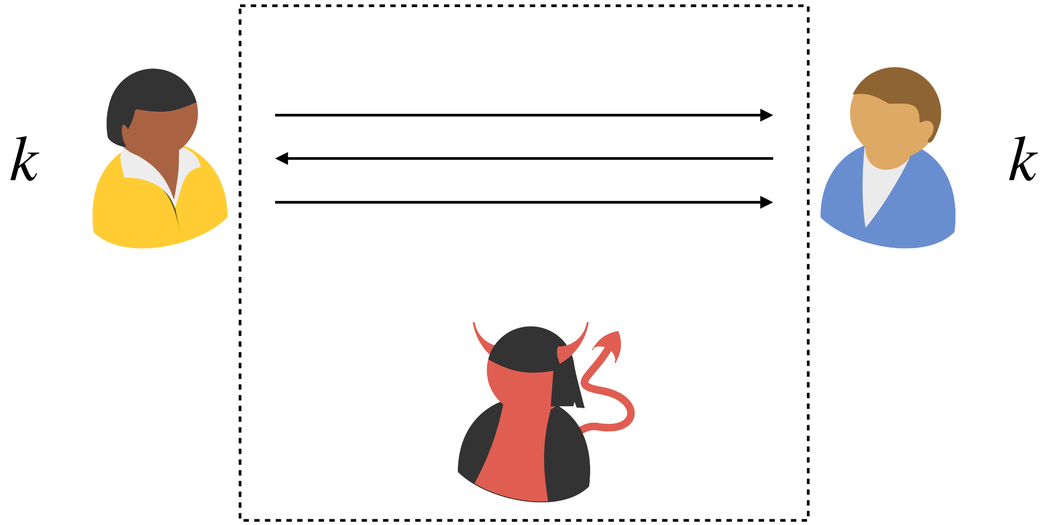


- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

- Security: given a *view* of a protocol (i.e. all the public messages), the agreed key should be indistinguishable from random.

$$x \xleftarrow{\$} \{0,\ldots,p-1\}$$

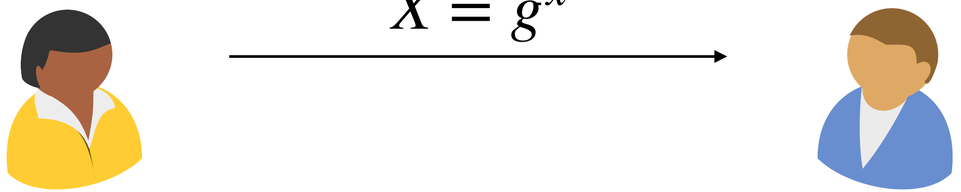
Key Exchange



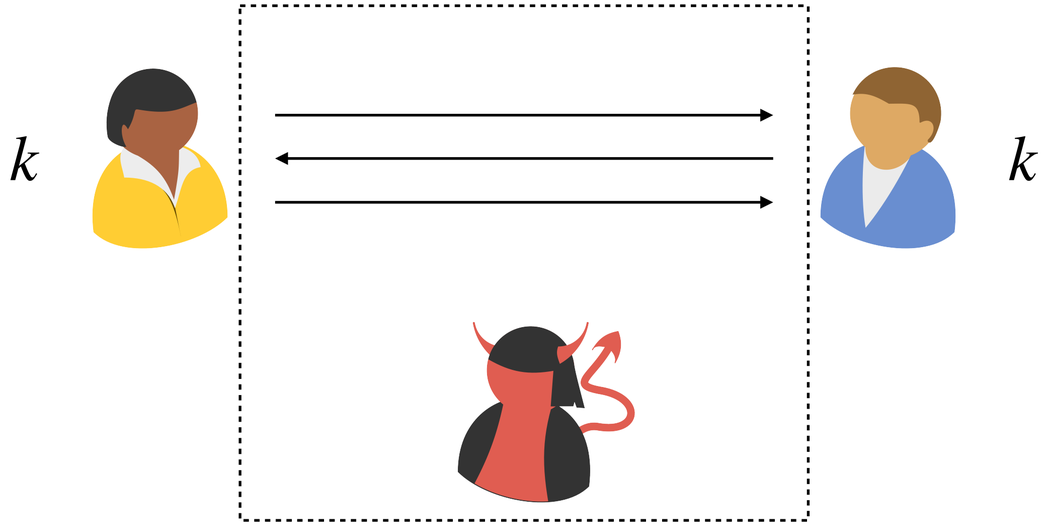
- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

- Security: given a *view* of a protocol (i.e. all the public messages), the agreed key should be indistinguishable from random.

$$x \xleftarrow{\$} \{0,\ldots,p-1\}$$



Key Exchange



- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

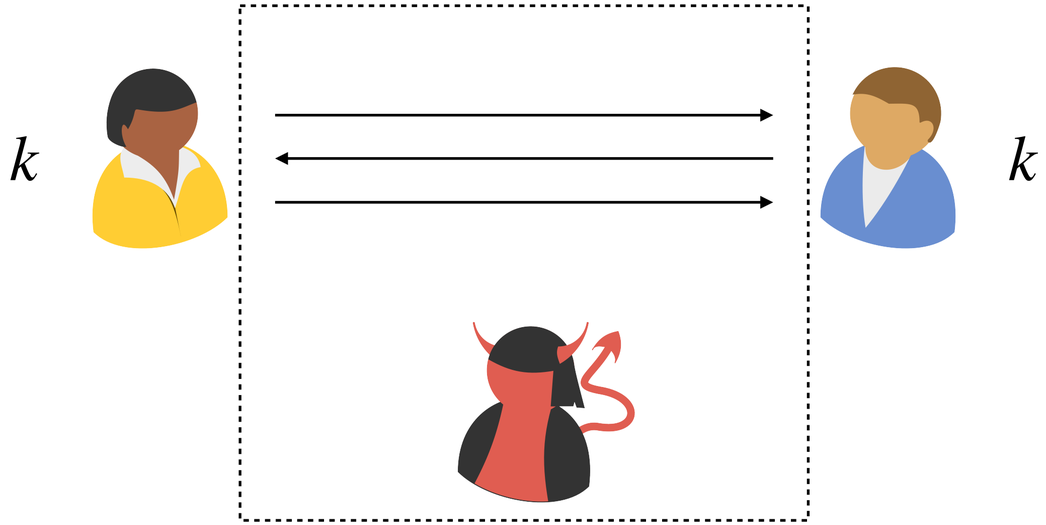
- Security: given a *view* of a protocol (i.e. all the public messages), the agreed key should be indistinguishable from random.

$$x \xleftarrow{\$} \{0,\ldots,p-1\}$$

$$X = g^x$$

$$y \xleftarrow{\$} \{0,\ldots,p-1\}$$

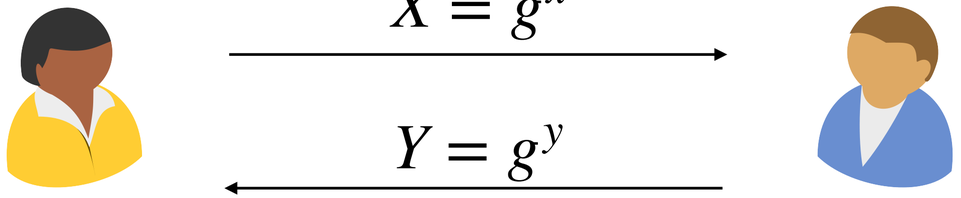
Key Exchange



- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

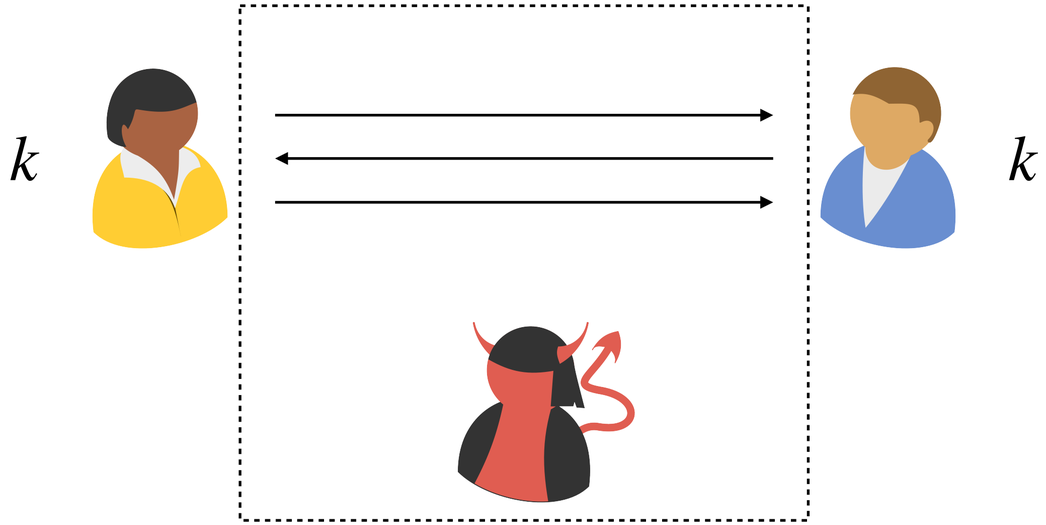
- Security: given a *view* of a protocol (i.e. all the public messages), the agreed key should be indistinguishable from random.

$$x \xleftarrow{\$} \{0,\ldots,p-1\}$$



$$y \xleftarrow{\$} \{0,\ldots,p-1\}$$

Key Exchange

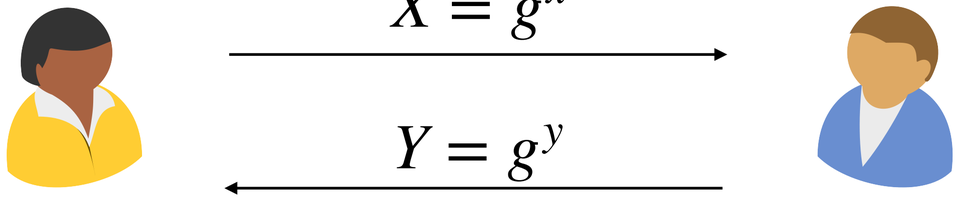


- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

- Security: given a *view* of a protocol (i.e. all the public messages), the agreed key should be indistinguishable from random.

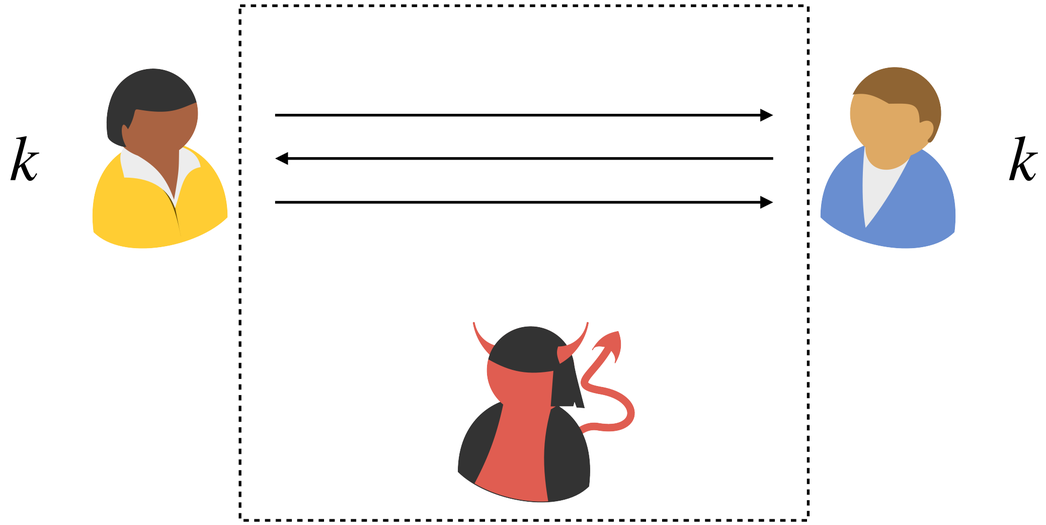
$$x \xleftarrow{\$} \{0,\ldots,p-1\}$$

$$k = Y^x$$



$$y \xleftarrow{\$} \{0,\ldots,p-1\}$$

Key Exchange

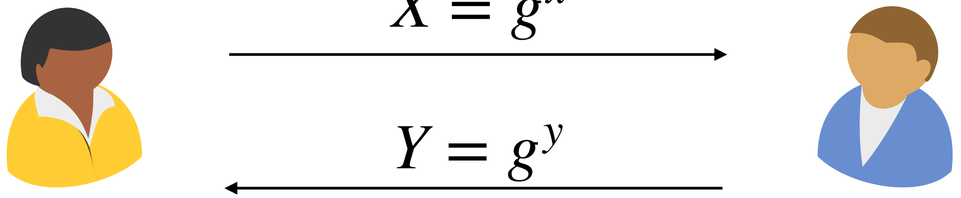


- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

- Security: given a *view* of a protocol (i.e. all the public messages), the agreed key should be indistinguishable from random.

$$x \xleftarrow{\$} \{0,\ldots,p-1\}$$

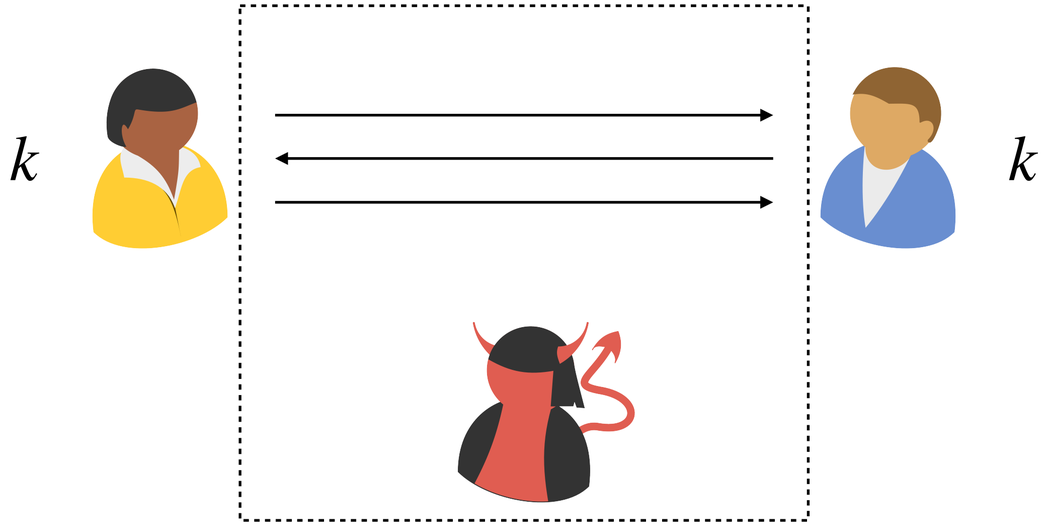
$$k = Y^x$$



$$y \xleftarrow{\$} \{0,\ldots,p-1\}$$

$$k = X^y$$

Key Exchange

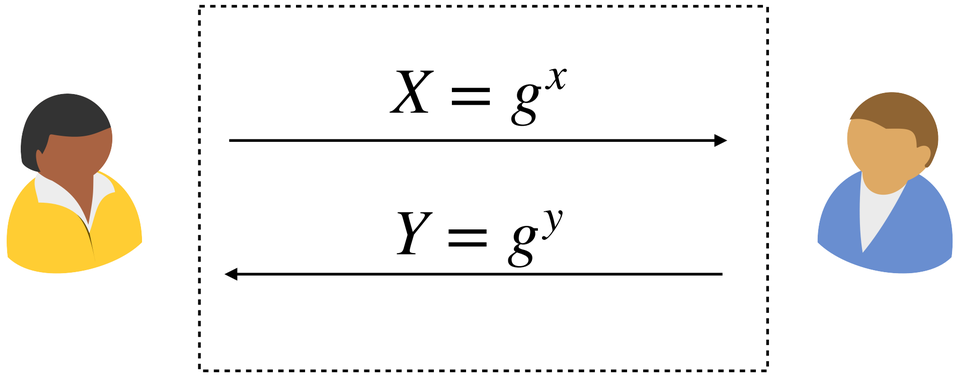


- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

- Security: given a *view* of a protocol (i.e. all the public messages), the agreed key should be indistinguishable from random.

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = Y^x$



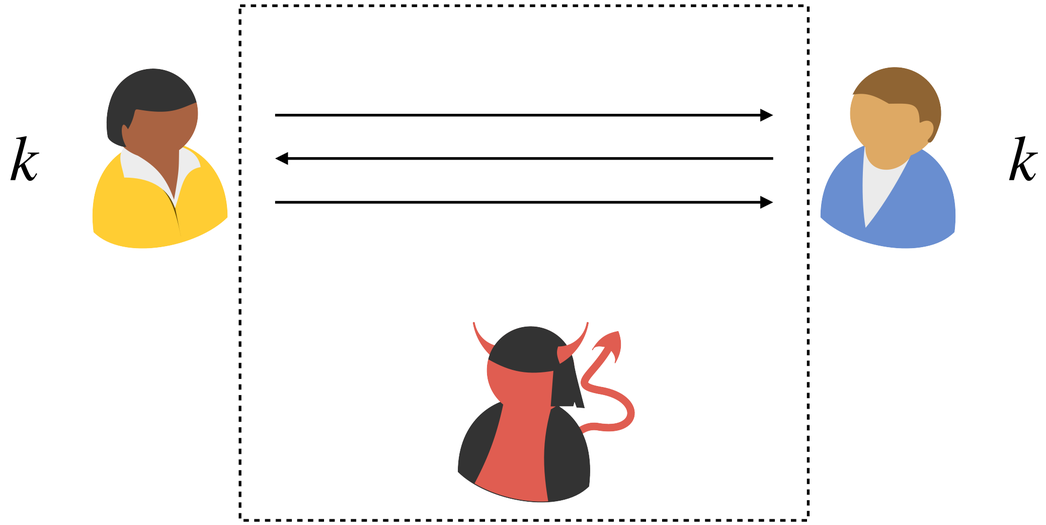
$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = X^y$

$\mathcal{A}$ sees g^x and g^y. The key is g^{xy}. By DDH k is indistinguishable from random!

Key Exchange

This protocol is known as "Diffie-Hellman Key Exchange"

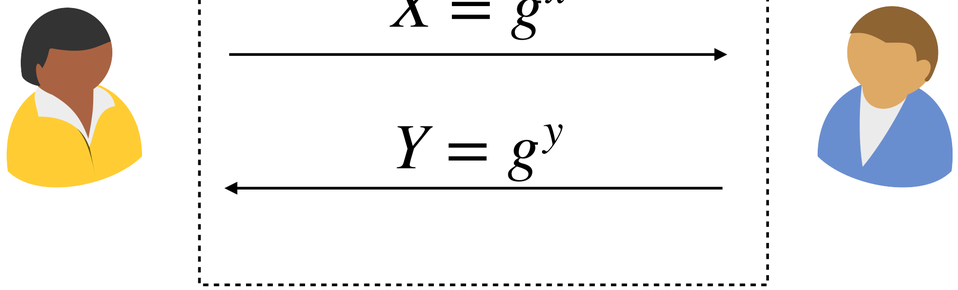


- We can avoid the problem of constructing a public-key encryption scheme if we find a way for Alice and Bob to *agree* on a key in public, without an adversary learning the key.

- Security: given a *view* of a protocol (i.e. all the public messages), the agreed key should be indistinguishable from random.

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = Y^x$



$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = X^y$

$\mathcal{A}$ sees g^x and g^y. The key is g^{xy}. By DDH k is indistinguishable from random!

ElGamal Encryption

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = Y^x$

$X = g^x$

$Y = g^y$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = X^y$

ElGamal Encryption

If Bob *published* his value g^y, Alice could do this protocol *non-interactively*

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = Y^x$

$X = g^x$

$Y = g^y$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = X^y$

ElGamal Encryption

If Bob *published* his value g^y, Alice could do this protocol *non-interactively*

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = Y^x$

$X = g^x$

$Y = g^y$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = X^y$

ElGamal Encryption

If Bob *published* his value g^y, Alice could do this protocol *non-interactively*

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = Y^x$

$X = g^x$

$Y = g^y$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = X^y$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

ElGamal Encryption

If Bob *published* his value g^y, Alice could do this protocol *non-interactively*

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = Y^x$

$X = g^x$

$Y = g^y$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = X^y$

$Y = g^y$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

ElGamal Encryption

If Bob *published* his value g^y, Alice could do this protocol *non-interactively*

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = Y^x$

$X = g^x$

$Y = g^y$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = X^y$

$Y = g^y$

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = Y^x$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

ElGamal Encryption

If Bob *published* his value g^y, Alice could do this protocol *non-interactively*

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = Y^x$

$X = g^x$

$Y = g^y$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = X^y$

$Y = g^y$

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$k = Y^x$

$X = g^x$

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

ElGamal Encryption

ElGamal Encryption

ElGamal Encryption

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

- KeyGen(1^λ):

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

- KeyGen(1^λ):
  - $y \xleftarrow{\$} \{0,\ldots,p-1\}$

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

- KeyGen(1^λ):
  - $y \xleftarrow{\$} \{0, \ldots, p-1\}$
  - $sk = y, pk = g^y$

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

- KeyGen(1^λ):
  - $y \xleftarrow{\$} \{0, \ldots, p-1\}$
  - $sk = y, pk = g^y$
- Enc(pk, m):

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

- KeyGen(1^λ):
  - $y \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $sk = y, pk = g^y$
- Enc(pk, m):
  - $x \xleftarrow{\$} \{0,1\}^\lambda$

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

- KeyGen(1^λ):
  - $y \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $sk = y, pk = g^y$
- Enc(pk, m):
  - $x \xleftarrow{\$} \{0,1\}^\lambda$
  - ct $:= (m * pk^x, g^x)$

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

- KeyGen(1^λ):
  - $y \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $sk = y, pk = g^y$
- Enc(pk, m):
  - $x \xleftarrow{\$} \{0,1\}^\lambda$
  - ct $:= (m * pk^x, g^x)$
- Dec$(k, (c_1, c_2))$:

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

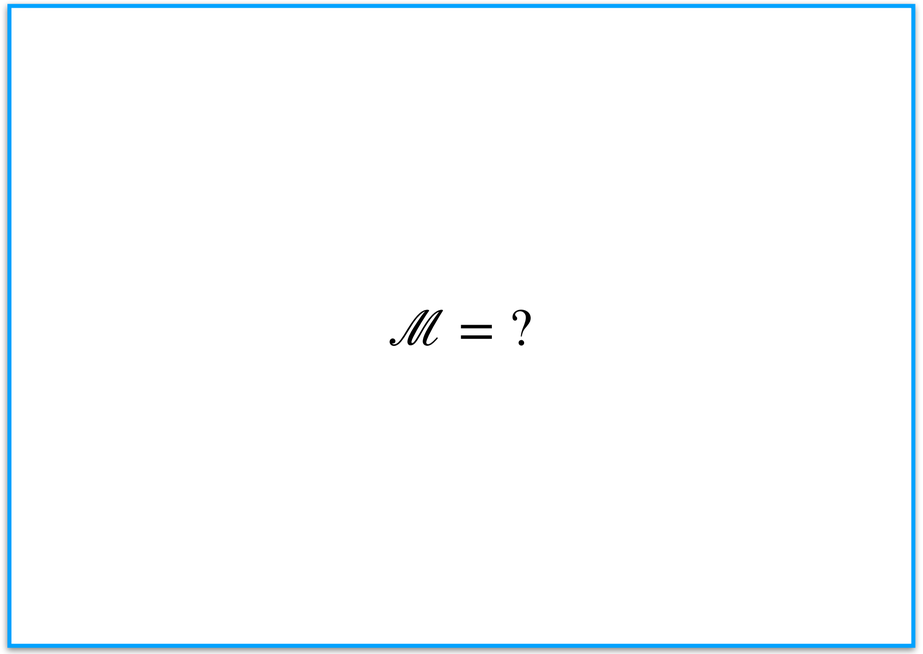
- KeyGen(1^λ):
  - $y \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $sk = y, pk = g^y$
- Enc(pk, m):
  - $x \xleftarrow{\$} \{0,1\}^\lambda$
  - ct $:= (m * pk^x, g^x)$
- Dec$(k, (c_1, c_2))$:
  - $m := (c_2^{sk})^{-1} * c_1.$

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

- KeyGen(1^λ):
  - $y \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $sk = y, pk = g^y$
- Enc(pk, m):
  - $x \xleftarrow{\$} \{0,1\}^\lambda$
  - ct $:= (m * pk^x, g^x)$
- Dec$(k, (c_1, c_2))$:
  - $m := (c_2^{sk})^{-1} * c_1$.



ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

- KeyGen(1^λ):
  - $y \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $sk = y, pk = g^y$
- Enc(pk, m):
  - $x \xleftarrow{\$} \{0,1\}^\lambda$
  - ct $:= (m * pk^x, g^x)$
- Dec$(k, (c_1, c_2))$:
  - $m := (c_2^{sk})^{-1} * c_1$.

$$\mathcal{M} = G$$

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

- KeyGen(1^λ):
  - $y \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $sk = y, pk = g^y$
- Enc(pk, m):
  - $x \xleftarrow{\$} \{0,1\}^\lambda$
  - ct $:= (m * pk^x, g^x)$
- Dec$(k, (c_1, c_2))$:
  - $m := (c_2^{sk})^{-1} * c_1$.

$$\mathscr{M} = G$$
$$\mathscr{K}_s = ?$$

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

- KeyGen(1^λ):
  - $y \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $sk = y, pk = g^y$
- Enc(pk, m):
  - $x \xleftarrow{\$} \{0,1\}^\lambda$
  - ct $:= (m * pk^x, g^x)$
- Dec$(k, (c_1, c_2))$:
  - $m := (c_2^{sk})^{-1} * c_1$.

$$\mathcal{M} = G$$
$$\mathcal{K}_s = \{0,\ldots,p-1\}$$

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

- KeyGen(1^λ):
  - $y \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $sk = y, pk = g^y$
- Enc(pk, m):
  - $x \xleftarrow{\$} \{0,1\}^\lambda$
  - ct $:= (m * pk^x, g^x)$
- Dec$(k, (c_1, c_2))$:
  - $m := (c_2^{sk})^{-1} * c_1$.

$$\mathcal{M} = G$$
$$\mathcal{K}_s = \{0,\ldots,p-1\}$$
$$\mathcal{C} = ?$$

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

- KeyGen(1^λ):
  - $y \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $sk = y, pk = g^y$
- Enc(pk, m):
  - $x \xleftarrow{\$} \{0,1\}^\lambda$
  - ct $:= (m * pk^x, g^x)$
- Dec$(k, (c_1, c_2))$:
  - $m := (c_2^{sk})^{-1} * c_1$.

$$\mathscr{M} = G$$
$$\mathscr{K}_s = \{0,\ldots,p-1\}$$
$$\mathscr{C} = G \times G$$

ElGamal Encryption

ElGamal Encryption

Let λ be the security parameter, and let $(G, *)$ be a group with generator g, order p, and where DDH holds.

- KeyGen(1^λ):
  - $y \xleftarrow{\$} \{0,\ldots,p-1\}$
  - $sk = y, pk = g^y$
- Enc(pk, m):
  - $x \xleftarrow{\$} \{0,1\}^\lambda$
  - ct $:= (m * pk^x, g^x)$
- Dec$(k, (c_1, c_2))$:
  - $m := (c_2^{sk})^{-1} * c_1$.

Correctness:

$$m := c_1 * (c_2^{sk})^{-1}$$
$$m := c_1 * ((g^x)^{sk})^{-1}$$
$$m := c_1 * ((g^x)^y)^{-1}$$
$$m := c_1 * g^{-xy}$$
$$m := m * pk^x * g^{-xy}$$
$$m := m * g^{xy} * g^{-xy}$$
$$m := m$$

ElGamal: Proof of Security

ElGamal: Proof of Security

$$\text{KeyGen}(1^\lambda): \; y \xleftarrow{\$} \{0,\ldots,p-1\}$$
$$sk := y, pk := g^y$$

$$\text{Enc}(\text{pk}, \text{m}): \; x \xleftarrow{\$} \{0,\ldots,p-1\}$$
$$\text{ct} := (m * pk^x, g^x)$$

ElGamal: Proof of Security

H_0

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$sk := y,\ pk := g^y$

pk →

(m_0, m_1) ←

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$\mathsf{ct} := (m_0 * pk^x, g^x)$

ct →

b' ←

$\mathsf{KeyGen}(1^\lambda)$: $y \xleftarrow{\$} \{0,\ldots,p-1\}$

$sk := y,\ pk := g^y$

$\mathsf{Enc}(\mathsf{pk}, \mathsf{m})$: $x \xleftarrow{\$} \{0,\ldots,p-1\}$

$\mathsf{ct} := (m * pk^x, g^x)$

ElGamal: Proof of Security

$$\text{KeyGen}(1^\lambda): \; y \xleftarrow{\$} \{0,\ldots,p-1\}, \; sk := y, pk := g^y$$

$$\text{Enc}(\text{pk}, \text{m}): \; x \xleftarrow{\$} \{0,\ldots,p-1\}, \; \text{ct} := (m * pk^x, g^x)$$

H_0

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$sk := y, pk := g^y$

pk →

(m_0, m_1) ←

ct →

b' ←

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$\text{ct} := (m_0 * pk^x, g^x)$

H_1

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$sk := y, pk := g^y$

pk →

(m_0, m_1) ←

ct →

b' ←

$r \xleftarrow{\$} \{0,\ldots,p-1\}$

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$\text{ct} := (m_0 * g^r, g^x)$

ElGamal: Proof of Security

H_0

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$sk := y,\ pk := g^y$

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$\mathsf{ct} := (m_0 * pk^x, g^x)$

pk →

(m_0, m_1) ←

ct →

b' ←

Prove $H_0 \overset{c}{\approx} H_1$ via a reduction to DDH

H_1

$y \xleftarrow{\$} \{0,\ldots,p-1\}$

$sk := y,\ pk := g^y$

$r \xleftarrow{\$} \{0,\ldots,p-1\}$

$x \xleftarrow{\$} \{0,\ldots,p-1\}$

$\mathsf{ct} := (m_0 * g^r, g^x)$

pk →

(m_0, m_1) ←

ct →

b' ←

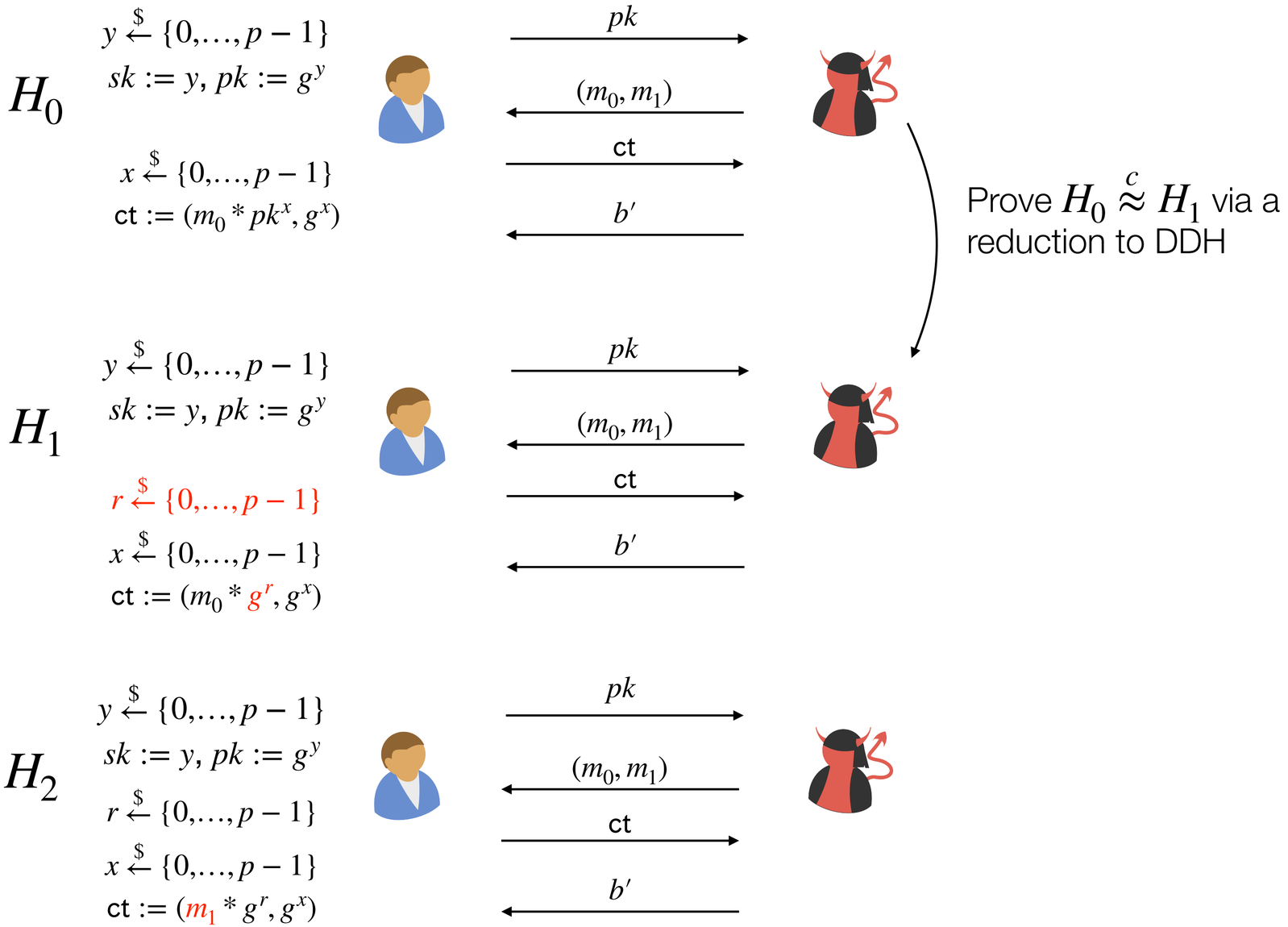
$\mathsf{KeyGen}(1^\lambda)$: $y \xleftarrow{\$} \{0,\ldots,p-1\}$, $sk := y,\ pk := g^y$

$\mathsf{Enc}(\mathsf{pk}, \mathsf{m})$: $x \xleftarrow{\$} \{0,\ldots,p-1\}$, $\mathsf{ct} := (m * pk^x, g^x)$

ElGamal: Proof of Security

$$\text{KeyGen}(1^\lambda): \; y \xleftarrow{\$} \{0,\ldots,p-1\}, \; sk := y, pk := g^y$$

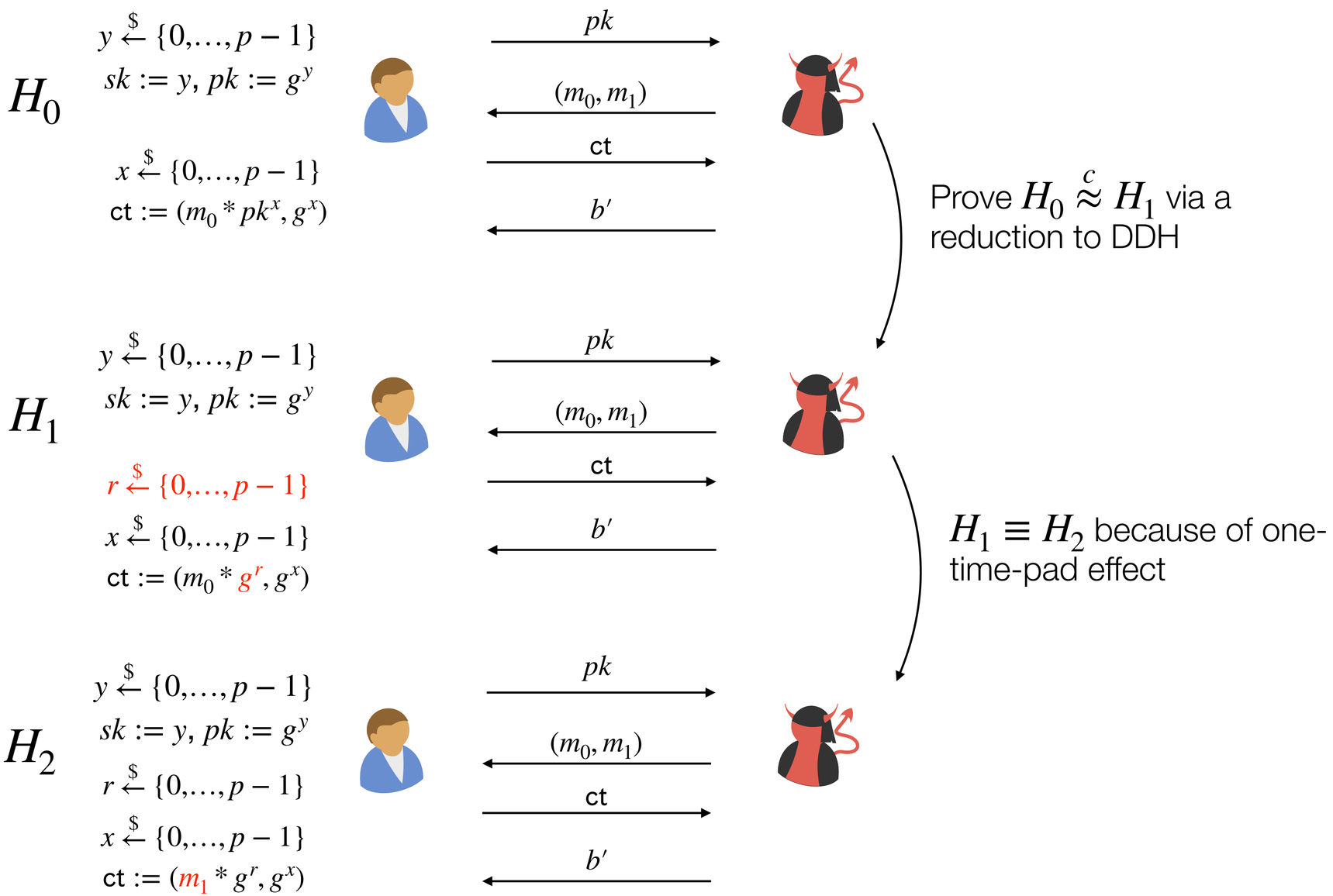
$$\text{Enc}(\text{pk}, \text{m}): \; x \xleftarrow{\$} \{0,\ldots,p-1\}, \; \text{ct} := (m * pk^x, g^x)$$



ElGamal: Proof of Security

KeyGen(1^λ) : $y \xleftarrow{\$} \{0,\ldots,p-1\}$, $sk := y, pk := g^y$

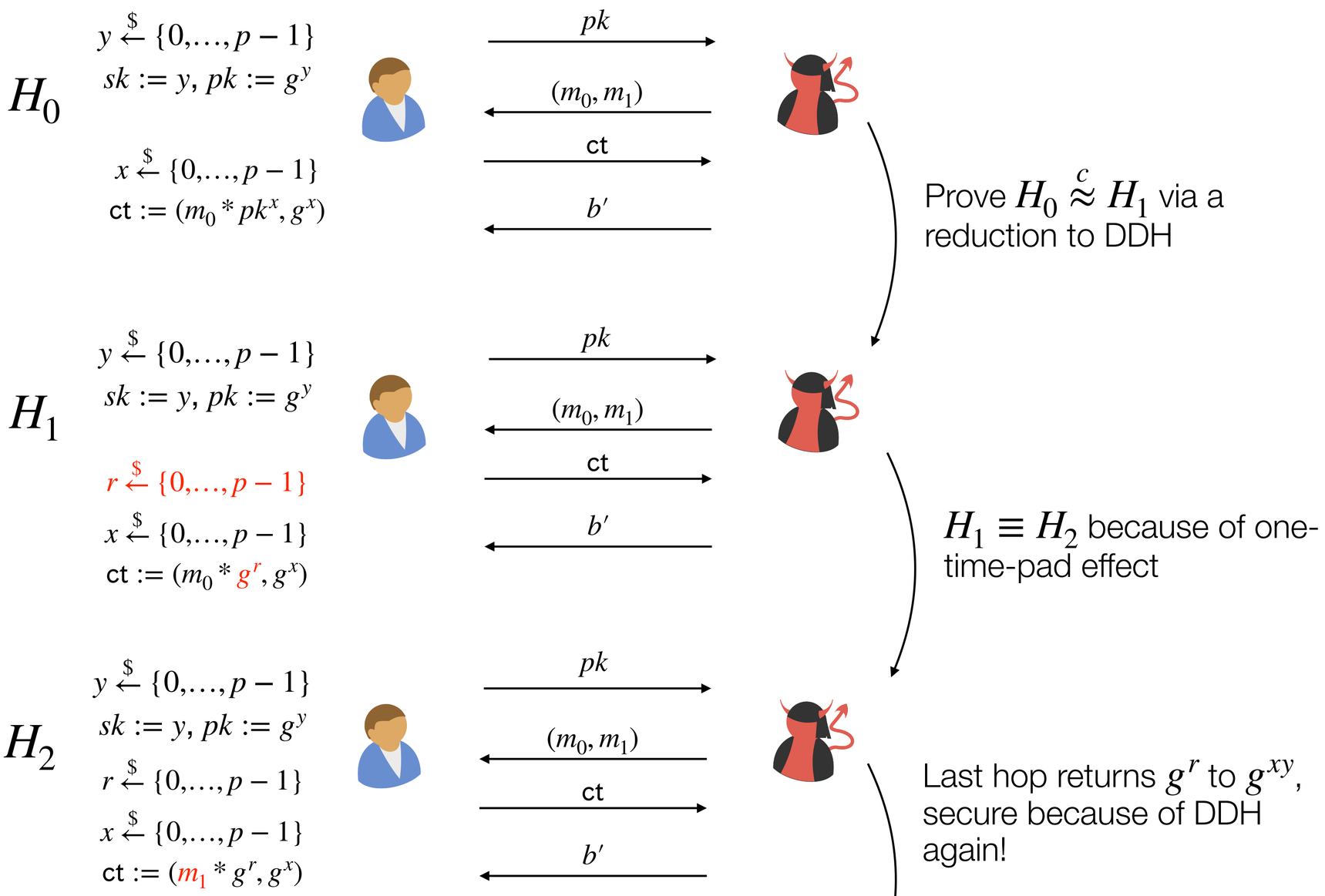
Enc(pk, m) : $x \xleftarrow{\$} \{0,\ldots,p-1\}$, $\text{ct} := (m * pk^x, g^x)$



ElGamal: Proof of Security

$$\text{KeyGen}(1^\lambda): y \xleftarrow{\$} \{0,\ldots,p-1\},\ sk := y,\ pk := g^y$$

$$\text{Enc}(\text{pk}, \text{m}): x \xleftarrow{\$} \{0,\ldots,p-1\},\ \text{ct} := (m * pk^x, g^x)$$



Generalizing ElGamal

Generalizing ElGamal

- We built ElGamal out of a way to do *non-interactive key agreement* (NIKE) using DDH.

Generalizing ElGamal

- We built ElGamal out of a way to do *non-interactive key agreement* (NIKE) using DDH.
- It turns out that this approach can be generalized!

Generalizing ElGamal

- We built ElGamal out of a way to do *non-interactive key agreement* (NIKE) using DDH.
- It turns out that this approach can be generalized!

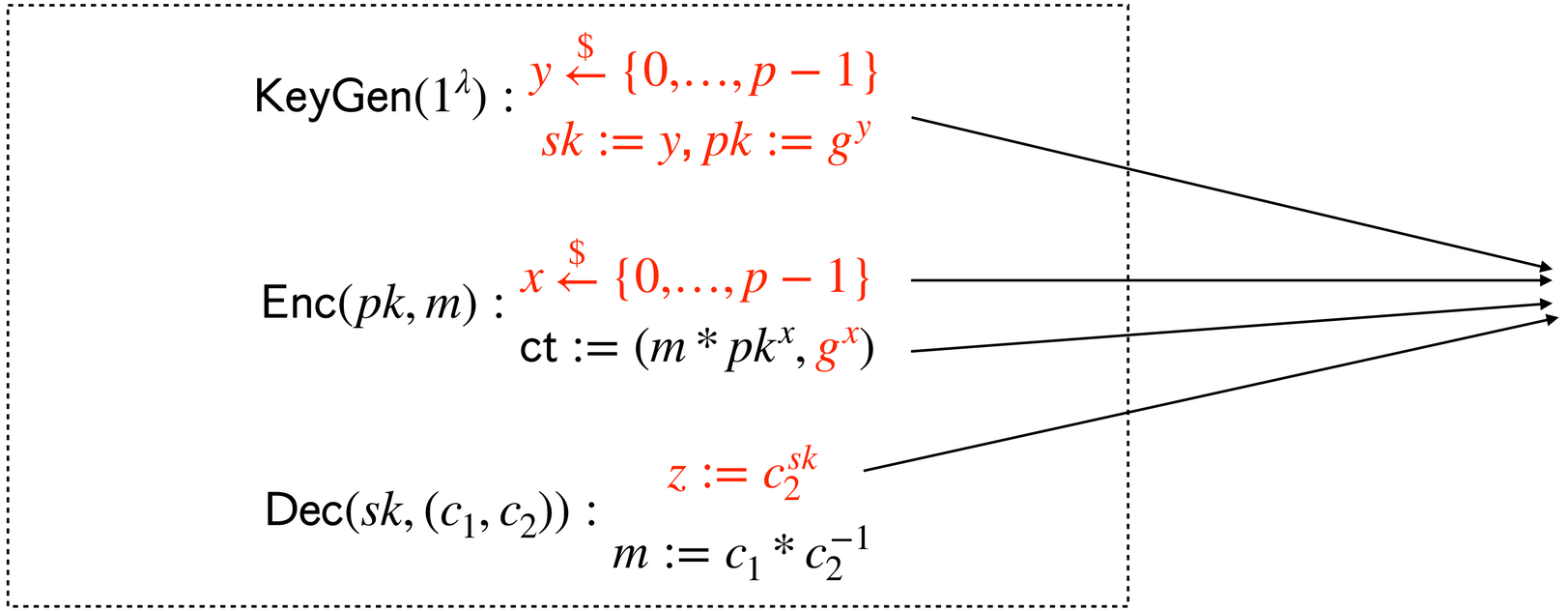
$$\mathsf{KeyGen}(1^\lambda): \begin{array}{c} y \xleftarrow{\$} \{0,\ldots,p-1\} \\ sk := y, pk := g^y \end{array}$$

$$\mathsf{Enc}(pk, m): \begin{array}{c} x \xleftarrow{\$} \{0,\ldots,p-1\} \\ \mathsf{ct} := (m * pk^x, g^x) \end{array}$$

$$\mathsf{Dec}(sk, (c_1, c_2)): \begin{array}{c} z := c_2^{sk} \\ m := c_1 * c_2^{-1} \end{array}$$

Generalizing ElGamal

- We built ElGamal out of a way to do *non-interactive key agreement* (NIKE) using DDH.
- It turns out that this approach can be generalized!



Ciphertext is one part Diffie-Hellman key exchange, and one part OTP with the shared key!

Non-Interactive Key Exchange (NIKE)

Non-Interactive Key Exchange Scheme Syntax

A *non-interactive key exchange scheme* consists of two (possibly probabilistic) algorithms:

Non-Interactive Key Exchange (NIKE)

Non-Interactive Key Exchange Scheme Syntax

A *non-interactive key exchange scheme* consists of two (possibly probabilistic) algorithms:

- KeyGen$(1^\lambda) \to (sk, pk)$ outputs a secret key $sk \in \mathcal{K}_s$ and a public key $pk \in \mathcal{K}_p$

Non-Interactive Key Exchange (NIKE)

Non-Interactive Key Exchange Scheme Syntax

A *non-interactive key exchange scheme* consists of two (possibly probabilistic) algorithms:

- KeyGen$(1^\lambda) \to (sk, pk)$ outputs a secret key $sk \in \mathcal{K}_s$ and a public key $pk \in \mathcal{K}_p$
- Derive$(sk_i, pk_j) \to k$ takes as input a secret key and a public key and outputs a shared key $k \in \mathcal{K}$

Non-Interactive Key Exchange (NIKE)

Non-Interactive Key Exchange Scheme Syntax

A *non-interactive key exchange scheme* consists of two (possibly probabilistic) algorithms:

- $\mathsf{KeyGen}(1^\lambda) \to (sk, pk)$ outputs a secret key $sk \in \mathscr{K}_s$ and a public key $pk \in \mathscr{K}_p$
- $\mathsf{Derive}(sk_i, pk_j) \to k$ takes as input a secret key and a public key and outputs a shared key $k \in \mathscr{K}$

Correctness: $$\Pr\left[\mathsf{Derive}(sk_0, pk_1) = \mathsf{Derive}(sk_1, pk_0) \;:\; \begin{matrix}(sk_0, pk_0) \leftarrow \mathsf{KeyGen}(\lambda) \\ (sk_1, pk_1) \leftarrow \mathsf{KeyGen}(\lambda)\end{matrix}\right] = 1$$

NIKE Security

NIKE Security

A non-interactive key exchange scheme (KeyGen, Derive) is *secure* if for all NUPPT $\mathscr{A}$, there exists a negligible function $\nu(\cdot)$ such that $\forall \lambda \in \mathbb{N}$

$$\Pr[\mathscr{A} \text{ wins GuessGame}] \leq \frac{1}{2} + \nu(\lambda)$$

NIKE Security

NIKE Security

A non-interactive key exchange scheme (KeyGen, Derive) is *secure* if for all NUPPT $\mathscr{A}$, there exists a negligible function $\nu(\cdot)$ such that $\forall \lambda \in \mathbb{N}$

$$\Pr[\mathscr{A} \text{ wins GuessGame}] \leq \frac{1}{2} + \nu(\lambda)$$

NIKE Security

NIKE Security

A non-interactive key exchange scheme (KeyGen, Derive) is *secure* if for all NUPPT $\mathscr{A}$, there exists a negligible function $\nu(\cdot)$ such that $\forall \lambda \in \mathbb{N}$

$$\Pr[\mathscr{A} \text{ wins GuessGame}] \leq \frac{1}{2} + \nu(\lambda)$$

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

NIKE Security

NIKE Security

A non-interactive key exchange scheme (KeyGen, Derive) is *secure* if for all NUPPT $\mathscr{A}$, there exists a negligible function $\nu(\cdot)$ such that $\forall \lambda \in \mathbb{N}$

$$\Pr[\mathscr{A} \text{ wins GuessGame}] \leq \frac{1}{2} + \nu(\lambda)$$

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

$$(sk_0, pk_0) \leftarrow \text{KeyGen}(1^\lambda)$$

$$(sk_1, pk_1) \leftarrow \text{KeyGen}(1^\lambda)$$

NIKE Security

NIKE Security

A non-interactive key exchange scheme (KeyGen, Derive) is *secure* if for all NUPPT $\mathscr{A}$, there exists a negligible function $\nu(\cdot)$ such that $\forall \lambda \in \mathbb{N}$

$$\Pr[\mathscr{A} \text{ wins GuessGame}] \leq \frac{1}{2} + \nu(\lambda)$$

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

$$(sk_0, pk_0) \leftarrow \text{KeyGen}(1^\lambda)$$

$$(sk_1, pk_1) \leftarrow \text{KeyGen}(1^\lambda)$$

$$k_0 := \text{Derive}(sk_0, pk_1)$$

$$k_1 \xleftarrow{\$} \mathscr{K}$$

NIKE Security

NIKE Security

A non-interactive key exchange scheme (KeyGen, Derive) is *secure* if for all NUPPT $\mathscr{A}$, there exists a negligible function $\nu(\cdot)$ such that $\forall \lambda \in \mathbb{N}$

$$\Pr[\mathscr{A} \text{ wins GuessGame}] \leq \frac{1}{2} + \nu(\lambda)$$

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

$$(sk_0, pk_0) \leftarrow \text{KeyGen}(1^\lambda)$$

$$(sk_1, pk_1) \leftarrow \text{KeyGen}(1^\lambda)$$

$$k_0 := \text{Derive}(sk_0, pk_1)$$

$$k_1 \xleftarrow{\$} \mathscr{K}$$

$$(pk_0, pk_1, k_b) \longrightarrow$$

NIKE Security

NIKE Security

A non-interactive key exchange scheme (KeyGen, Derive) is *secure* if for all NUPPT $\mathscr{A}$, there exists a negligible function $\nu(\cdot)$ such that $\forall \lambda \in \mathbb{N}$

$$\Pr[\mathscr{A} \text{ wins GuessGame}] \leq \frac{1}{2} + \nu(\lambda)$$

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

$(sk_0, pk_0) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk_1, pk_1) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k_0 := \mathsf{Derive}(sk_0, pk_1)$

$k_1 \xleftarrow{\$} \mathscr{K}$

(pk_0, pk_1, k_b)

b'

NIKE Security

NIKE Security

A non-interactive key exchange scheme (KeyGen, Derive) is *secure* if for all NUPPT $\mathscr{A}$, there exists a negligible function $\nu(\cdot)$ such that $\forall \lambda \in \mathbb{N}$

$$\Pr[\mathscr{A} \text{ wins GuessGame}] \leq \frac{1}{2} + \nu(\lambda)$$

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

$(sk_0, pk_0) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk_1, pk_1) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k_0 := \mathsf{Derive}(sk_0, pk_1)$

$k_1 \xleftarrow{\$} \mathscr{K}$

(pk_0, pk_1, k_b) →

← b'

Wins if $b' = b$

NIKE Security

NIKE Security

A non-interactive key exchange scheme (KeyGen, Derive) is *secure* if for all NUPPT $\mathscr{A}$, there exists a negligible function $\nu(\cdot)$ such that $\forall \lambda \in \mathbb{N}$

$$\left| \Pr[\mathscr{A} \text{ outputs } 1 \text{ in Game}_0] - \Pr[\mathscr{A} \text{ outputs } 1 \text{ in Game}_1] \right| \leq \nu(\lambda)$$

Game_0

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

$(sk_0, pk_0) \leftarrow \text{KeyGen}(1^\lambda)$

$(sk_1, pk_1) \leftarrow \text{KeyGen}(1^\lambda)$

$k := \text{Derive}(sk_0, pk_1)$

(pk_0, pk_1, k) →

← b'

Game_1

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

$(sk_0, pk_0) \leftarrow \text{KeyGen}(1^\lambda)$

$(sk_1, pk_1) \leftarrow \text{KeyGen}(1^\lambda)$

$k \xleftarrow{\$} \mathscr{K}$

(pk_0, pk_1, k) →

← b'

NIKE Encryption

NIKE Encryption

NIKE Encryption

NIKE Encryption

NIKE Encryption

Let (KeyGen, Derive) be a NIKE scheme for $(\mathcal{K}_s, \mathcal{K}_p, \mathcal{K})$, and (Enc, Dec) be an IND-CPA secure secret-key encryption scheme with key space $\mathcal{K}$.

NIKE Encryption

NIKE Encryption

Let (KeyGen, Derive) be a NIKE scheme for $(\mathcal{K}_s, \mathcal{K}_p, \mathcal{K})$, and (Enc, Dec) be an IND-CPA secure secret-key encryption scheme with key space $\mathcal{K}$.

- $\mathsf{KeyGen}'(1^\lambda)$:

NIKE Encryption

NIKE Encryption

Let (KeyGen, Derive) be a NIKE scheme for $(\mathcal{K}_s, \mathcal{K}_p, \mathcal{K})$, and (Enc, Dec) be an IND-CPA secure secret-key encryption scheme with key space $\mathcal{K}$.

- $\text{KeyGen}'(1^\lambda)$:
  - $(sk, pk) \leftarrow \text{KeyGen}(1^\lambda)$

NIKE Encryption

NIKE Encryption

Let (KeyGen, Derive) be a NIKE scheme for $(\mathcal{K}_s, \mathcal{K}_p, \mathcal{K})$, and (Enc, Dec) be an IND-CPA secure secret-key encryption scheme with key space $\mathcal{K}$.

- $\text{KeyGen}'(1^\lambda)$:
  - $(sk, pk) \leftarrow \text{KeyGen}(1^\lambda)$
- $\text{Enc}'(pk, m)$:

NIKE Encryption

NIKE Encryption

Let (KeyGen, Derive) be a NIKE scheme for $(\mathcal{K}_s, \mathcal{K}_p, \mathcal{K})$, and (Enc, Dec) be an IND-CPA secure secret-key encryption scheme with key space $\mathcal{K}$.

- $\mathsf{KeyGen}'(1^\lambda)$:
  - $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
- $\mathsf{Enc}'(pk, m)$:
  - $(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

NIKE Encryption

NIKE Encryption

Let (KeyGen, Derive) be a NIKE scheme for $(\mathcal{K}_s, \mathcal{K}_p, \mathcal{K})$, and (Enc, Dec) be an IND-CPA secure secret-key encryption scheme with key space $\mathcal{K}$.

- $\mathsf{KeyGen}'(1^\lambda)$:
  - $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
- $\mathsf{Enc}'(pk, m)$:
  - $(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$
  - $k := \mathsf{Derive}(sk', pk)$

NIKE Encryption

NIKE Encryption

Let (KeyGen, Derive) be a NIKE scheme for $(\mathcal{K}_s, \mathcal{K}_p, \mathcal{K})$, and (Enc, Dec) be an IND-CPA secure secret-key encryption scheme with key space $\mathcal{K}$.

- $\mathsf{KeyGen}'(1^\lambda)$:
  - $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
- $\mathsf{Enc}'(pk, m)$:
  - $(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$
  - $k := \mathsf{Derive}(sk', pk)$
  - $c \leftarrow \mathsf{Enc}(k, m)$

NIKE Encryption

NIKE Encryption

Let (KeyGen, Derive) be a NIKE scheme for $(\mathcal{K}_s, \mathcal{K}_p, \mathcal{K})$, and (Enc, Dec) be an IND-CPA secure secret-key encryption scheme with key space $\mathcal{K}$.

- $\mathsf{KeyGen}'(1^\lambda)$:
  - $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
- $\mathsf{Enc}'(pk, m)$:
  - $(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$
  - $k := \mathsf{Derive}(sk', pk)$
  - $c \leftarrow \mathsf{Enc}(k, m)$
  - $\mathsf{ct} := (c, pk')$

NIKE Encryption

NIKE Encryption

Let (KeyGen, Derive) be a NIKE scheme for $(\mathcal{K}_s, \mathcal{K}_p, \mathcal{K})$, and (Enc, Dec) be an IND-CPA secure secret-key encryption scheme with key space $\mathcal{K}$.

- $\mathsf{KeyGen}'(1^\lambda)$:
  - $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
- $\mathsf{Enc}'(pk, m)$:
  - $(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$
  - $k := \mathsf{Derive}(sk', pk)$
  - $c \leftarrow \mathsf{Enc}(k, m)$
  - $\mathsf{ct} := (c, pk')$
- $\mathsf{Dec}'(sk, (c_1, c_2))$:

NIKE Encryption

NIKE Encryption

Let (KeyGen, Derive) be a NIKE scheme for $(\mathcal{K}_s, \mathcal{K}_p, \mathcal{K})$, and (Enc, Dec) be an IND-CPA secure secret-key encryption scheme with key space $\mathcal{K}$.

- $\mathsf{KeyGen}'(1^\lambda)$:
  - $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
- $\mathsf{Enc}'(pk, m)$:
  - $(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$
  - $k := \mathsf{Derive}(sk', pk)$
  - $c \leftarrow \mathsf{Enc}(k, m)$
  - $\mathsf{ct} := (c, pk')$
- $\mathsf{Dec}'(sk, (c_1, c_2))$:
  - $k := \mathsf{Derive}(sk, c_2)$

NIKE Encryption

NIKE Encryption

Let (KeyGen, Derive) be a NIKE scheme for $(\mathcal{K}_s, \mathcal{K}_p, \mathcal{K})$, and (Enc, Dec) be an IND-CPA secure secret-key encryption scheme with key space $\mathcal{K}$.

- $\mathsf{KeyGen}'(1^\lambda)$:
  - $(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$
- $\mathsf{Enc}'(pk, m)$:
  - $(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$
  - $k := \mathsf{Derive}(sk', pk)$
  - $c \leftarrow \mathsf{Enc}(k, m)$
  - $\mathsf{ct} := (c, pk')$
- $\mathsf{Dec}'(sk, (c_1, c_2))$:
  - $k := \mathsf{Derive}(sk, c_2)$
  - $m := \mathsf{Dec}(k, c_1)$

NIKE Encryption: Proof of Security

NIKE Encryption: Proof of Security

H_0

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k := \mathsf{Derive}(sk', pk)$

$c \leftarrow \mathsf{Enc}(k, m_0)$

$\mathsf{ct} := (c, pk')$

pk →

← (m_0, m_1)

ct →

← b'

NIKE Encryption: Proof of Security

H_0

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k := \mathsf{Derive}(sk', pk)$

$c \leftarrow \mathsf{Enc}(k, m_0)$

$\mathsf{ct} := (c, pk')$

pk →

(m_0, m_1) ←

ct →

b' ←

H_1

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k \xleftarrow{\$} \mathcal{K}$

$c \leftarrow \mathsf{Enc}(k, m_0)$

$\mathsf{ct} := (c, pk')$

pk →

(m_0, m_1) ←

ct →

b' ←

NIKE Encryption: Proof of Security

H_0

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k := \mathsf{Derive}(sk', pk)$

$c \leftarrow \mathsf{Enc}(k, m_0)$

$\mathsf{ct} := (c, pk')$

pk →

← (m_0, m_1)

ct →

← b'

Prove $H_0 \overset{c}{\approx} H_1$ via a reduction NIKE security

H_1

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k \xleftarrow{\$} \mathscr{K}$

$c \leftarrow \mathsf{Enc}(k, m_0)$

$\mathsf{ct} := (c, pk')$

pk →

← (m_0, m_1)

ct →

← b'

NIKE Encryption: Proof of Security

H_0

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k := \mathsf{Derive}(sk', pk)$

$c \leftarrow \mathsf{Enc}(k, m_0)$

$\mathsf{ct} := (c, pk')$

pk →

← (m_0, m_1)

ct →

← b'

Prove $H_0 \overset{c}{\approx} H_1$ via a reduction NIKE security

H_1

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k \xleftarrow{\$} \mathscr{K}$

$c \leftarrow \mathsf{Enc}(k, m_0)$

$\mathsf{ct} := (c, pk')$

pk →

← (m_0, m_1)

ct →

← b'

H_2

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k \xleftarrow{\$} \mathscr{K}$

$c \leftarrow \mathsf{Enc}(k, m_1)$

$\mathsf{ct} := (c, pk')$

pk →

← (m_0, m_1)

ct →

← b'

NIKE Encryption: Proof of Security

H_0

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k := \mathsf{Derive}(sk', pk)$

$c \leftarrow \mathsf{Enc}(k, m_0)$

$\mathsf{ct} := (c, pk')$

pk →

← (m_0, m_1)

ct →

← b'

Prove $H_0 \overset{c}{\approx} H_1$ via a reduction NIKE security

H_1

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k \xleftarrow{\$} \mathscr{K}$

$c \leftarrow \mathsf{Enc}(k, m_0)$

$\mathsf{ct} := (c, pk')$

pk →

← (m_0, m_1)

ct →

← b'

Prove $H_1 \overset{c}{\approx} H_2$ via a reduction to IND-CPA security of encryption scheme

H_2

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k \xleftarrow{\$} \mathscr{K}$

$c \leftarrow \mathsf{Enc}(k, m_1)$

$\mathsf{ct} := (c, pk')$

pk →

← (m_0, m_1)

ct →

← b'

NIKE Encryption: Proof of Security

H_0

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k := \mathsf{Derive}(sk', pk)$

$c \leftarrow \mathsf{Enc}(k, m_0)$

$\mathsf{ct} := (c, pk')$

pk →

← (m_0, m_1)

ct →

← b'

Prove $H_0 \overset{c}{\approx} H_1$ via a reduction NIKE security

H_1

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k \overset{\$}{\leftarrow} \mathscr{K}$

$c \leftarrow \mathsf{Enc}(k, m_0)$

$\mathsf{ct} := (c, pk')$

pk →

← (m_0, m_1)

ct →

← b'

Prove $H_1 \overset{c}{\approx} H_2$ via a reduction to IND-CPA security of encryption scheme

H_2

$(sk, pk) \leftarrow \mathsf{KeyGen}(1^\lambda)$

$(sk', pk') \leftarrow \mathsf{KeyGen}(1^\lambda)$

$k \overset{\$}{\leftarrow} \mathscr{K}$

$c \leftarrow \mathsf{Enc}(k, m_1)$

$\mathsf{ct} := (c, pk')$

pk →

← (m_0, m_1)

ct →

← b'

Last hop returns k to $\mathsf{Derive}(sk', pk)$, secure because of NIKE security again!

Analyzing Assumptions: Discrete Log

Discrete Log:
For a group $(G, *)$ of order p,
given g^a for a random
$a \in \{0,\ldots,p-1\}$, find a

Analyzing Assumptions: Discrete Log

Discrete Log:
For a group $(G, *)$ of order p,
given g^a for a random
$a \in \{0,\ldots,p-1\}$, find a

- How do we know that discrete log is hard?

Analyzing Assumptions: Discrete Log

Discrete Log:
For a group $(G, *)$ of order p,
given g^a for a random
$a \in \{0,\ldots,p-1\}$, find a

- How do we know that discrete log is hard?
- Easy place to start: brute force attacks:

Analyzing Assumptions: Discrete Log

Discrete Log:
For a group $(G, *)$ of order p,
given g^a for a random
$a \in \{0,\ldots,p-1\}$, find a

- How do we know that discrete log is hard?
- Easy place to start: brute force attacks:
  - G is of order p. If p is small (say $p \in O(\lambda)$) we can just try all possible values of a.

Analyzing Assumptions: Discrete Log

Discrete Log:
For a group $(G, *)$ of order p,
given g^a for a random
$a \in \{0,\ldots,p-1\}$, find a

- How do we know that discrete log is hard?
- Easy place to start: brute force attacks:
  - G is of order p. If p is small (say $p \in O(\lambda)$) we can just try all possible values of a.
  - Key question: can we do any better? Brute-force attack runs in $O(p)$.

Analyzing Assumptions: Discrete Log

Discrete Log:
For a group $(G, *)$ of order p,
given g^a for a random
$a \in \{0,\ldots,p-1\}$, find a

- How do we know that discrete log is hard?
- Easy place to start: brute force attacks:
  - G is of order p. If p is small (say $p \in O(\lambda)$) we can just try all possible values of a.
  - Key question: can we do any better? Brute-force attack runs in $O(p)$.
  - Cryptanalysis: the science of developing and analyzing attacks on cryptographic assumptions.

Analyzing Assumptions: Discrete Log

Discrete Log:
For a group $(G, *)$ of order p,
given g^a for a random
$a \in \{0,\ldots,p-1\}$, find a

- How do we know that discrete log is hard?
- Easy place to start: brute force attacks:
  - G is of order p. If p is small (say $p \in O(\lambda)$) we can just try all possible values of a.
  - Key question: can we do any better? Brute-force attack runs in $O(p)$.
  - Cryptanalysis: the science of developing and analyzing attacks on cryptographic assumptions.
  - “Baby-step giant step:” an algorithm for computing discrete log that runs in $O(\sqrt{p})$

Analyzing Assumptions: Discrete Log

Discrete Log:
For a group $(G, *)$ of order p,
given g^a for a random
$a \in \{0,\ldots,p-1\}$, find a

- How do we know that discrete log is hard?
- Easy place to start: brute force attacks:
  - G is of order p. If p is small (say $p \in O(\lambda)$) we can just try all possible values of a.
  - Key question: can we do any better? Brute-force attack runs in $O(p)$.
  - Cryptanalysis: the science of developing and analyzing attacks on cryptographic assumptions.
  - "Baby-step giant step:" an algorithm for computing discrete log that runs in $O(\sqrt{p})$
    - To get "128 bits of security" (i.e. an assurance that an adversary attacking the scheme successfully must run in time 2^{128}) we need to use groups of order $\geq 2^{256}$!

Usage in practice: Discrete Log Assumptions

Elliptic curve picture from: https://www.globalsign.com/en/blog/elliptic-curve-cryptography

Usage in practice: Discrete Log Assumptions

- We use a lot of discrete-log based crypto systems (i.e. things based on DDH or CDH) today

Elliptic curve picture from: https://www.globalsign.com/en/blog/elliptic-curve-cryptography

Usage in practice: Discrete Log Assumptions

- We use a lot of discrete-log based crypto systems (i.e. things based on DDH or CDH) today
- Most TLS connections use a key-exchange based on Diffie-Hellman!

Elliptic curve picture from: https://www.globalsign.com/en/blog/elliptic-curve-cryptography

Usage in practice: Discrete Log Assumptions

- We use a lot of discrete-log based crypto systems (i.e. things based on DDH or CDH) today
- Most TLS connections use a key-exchange based on Diffie-Hellman!
- To actually deploy something we need to pick a specific group:

Elliptic curve picture from: https://www.globalsign.com/en/blog/elliptic-curve-cryptography

Usage in practice: Discrete Log Assumptions

- We use a lot of discrete-log based crypto systems (i.e. things based on DDH or CDH) today
- Most TLS connections use a key-exchange based on Diffie-Hellman!
- To actually deploy something we need to pick a specific group:
- Back in the day: $G = \{1, \ldots, p\}$, $* = \times$

Elliptic curve picture from: https://www.globalsign.com/en/blog/elliptic-curve-cryptography

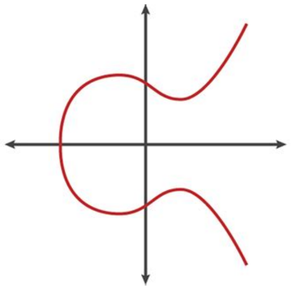
Usage in practice: Discrete Log Assumptions

- We use a lot of discrete-log based crypto systems (i.e. things based on DDH or CDH) today
- Most TLS connections use a key-exchange based on Diffie-Hellman!
- To actually deploy something we need to pick a specific group:
- Back in the day: $G = \{1, \ldots, p\}$, $* = \times$
  - Group is the group of integers modulo a large prime p with multiplication mod p as the operation.

Elliptic curve picture from: https://www.globalsign.com/en/blog/elliptic-curve-cryptography

Usage in practice: Discrete Log Assumptions

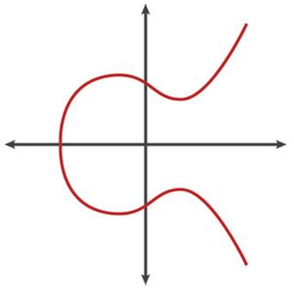
- We use a lot of discrete-log based crypto systems (i.e. things based on DDH or CDH) today
- Most TLS connections use a key-exchange based on Diffie-Hellman!
- To actually deploy something we need to pick a specific group:
- Back in the day: $G = \{1, \ldots, p\}$, $* = \times$
  - Group is the group of integers modulo a large prime p with multiplication mod p as the operation.
- Modern Era: G is the set of points on an *elliptic curve*.



Elliptic curve picture from: https://www.globalsign.com/en/blog/elliptic-curve-cryptography

Usage in practice: Discrete Log Assumptions

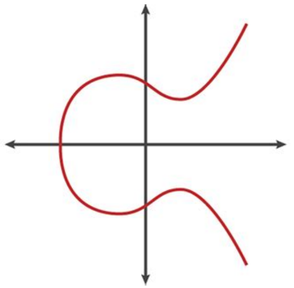
- We use a lot of discrete-log based crypto systems (i.e. things based on DDH or CDH) today
- Most TLS connections use a key-exchange based on Diffie-Hellman!
- To actually deploy something we need to pick a specific group:
- Back in the day: $G = \{1, \ldots, p\}$, $* = \times$
  - Group is the group of integers modulo a large prime p with multiplication mod p as the operation.
- Modern Era: G is the set of points on an *elliptic curve*.
  - Popular curves: X25519, P-256



Elliptic curve picture from: https://www.globalsign.com/en/blog/elliptic-curve-cryptography

Usage in practice: Discrete Log Assumptions

- We use a lot of discrete-log based crypto systems (i.e. things based on DDH or CDH) today
- Most TLS connections use a key-exchange based on Diffie-Hellman!
- To actually deploy something we need to pick a specific group:
- Back in the day: $G = \{1, \ldots, p\}$, $* = \times$
  - Group is the group of integers modulo a large prime p with multiplication mod p as the operation.
- Modern Era: G is the set of points on an *elliptic curve*.
  - Popular curves: X25519, P-256
  - Algorithm known as “Elliptic Curve Diffie-Hellman” (ECDH)



Elliptic curve picture from: https://www.globalsign.com/en/blog/elliptic-curve-cryptography