Key Exchange

Elena Kirshanova

Course “Information and Network Security”
Lecture 8
28 апреля 2020 г.

We've studied so far

- Pseudo random generators
- Stream Ciphers
- Block Ciphers
- Message Authentication Codes
- Hash functions
- Authenticated Encryption

We've studied so far

- Pseudo random generators
- Stream Ciphers
- Block Ciphers
- Message Authentication Codes
- Hash functions
- Authenticated Encryption

Except PRGs, all these primitives require a shared key.

Where does this key come from?

We've studied so far

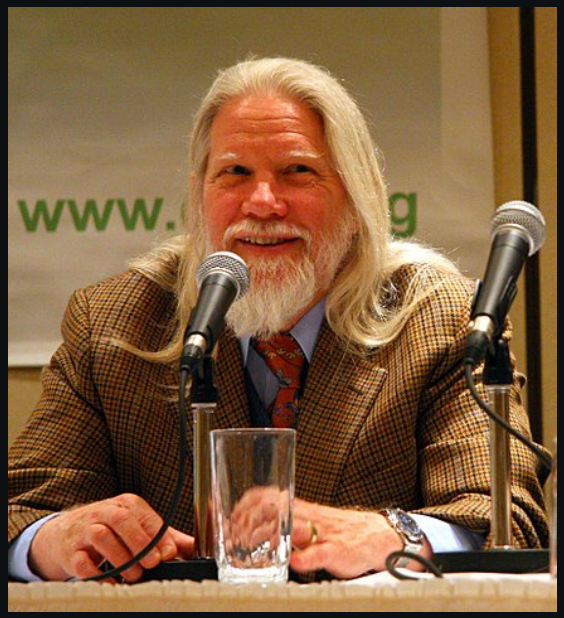
- Pseudo random generators
- Stream Ciphers
- Block Ciphers
- Message Authentication Codes
- Hash functions
- Authenticated Encryption

Except PRGs, all these primitives require a shared key.

Where does this key come from?

Use public key crypto!

Diffie-Hellman Key Exchange



Whitfield Diffie

Martin Hellman

©Wikipedia

- published by Whitfield Diffie and Martin Hellman in 1976
- largely used in modern Internet protocols (TLS)
- relies on hardness of the discrete logarithm problem

Math crash course I: modulo arithmetic

Let p be a large prime (approx. 2000bits)

- $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ – finite field

Math crash course I: modulo arithmetic

Let p be a large prime (approx. 2000bits)

- $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ – finite field
- elements in $\mathbb{Z}_p$ are added and multiplied modulo p, i.e., for $x, y \in \mathbb{Z}_p$

$$x + y \bmod p = \mathrm{rem}(x + y, p)$$
$$x \cdot y \bmod p = \mathrm{rem}(x \cdot y, p)$$

rem – remained (остаток от целочисл. деления)

Math crash course I: modulo arithmetic

Let p be a large prime (approx. 2000bits)

- $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ – finite field
- elements in $\mathbb{Z}_p$ are added and multiplied modulo p, i.e., for $x, y \in \mathbb{Z}_p$

$$x + y \bmod p = \mathrm{rem}(x + y, p)$$
$$x \cdot y \bmod p = \mathrm{rem}(x \cdot y, p)$$

rem – remained (остаток от целочисл. деления)
Ex.: $p = 7, \mathbb{Z}_p = \{0, 1, 2, 3, 4, 5, 6\}$

$$5 + 6 \bmod p = \mathrm{rem}(11, 7) = 4$$
$$3 \cdot 3 \bmod p = \mathrm{rem}(9, 7) = 2$$

Math crash course I: modulo arithmetic

Let p be a large prime (approx. 2000bits)

- $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ – finite field
- elements in $\mathbb{Z}_p$ are added and multiplied modulo p, i.e., for $x, y \in \mathbb{Z}_p$

$$x + y \bmod p = \mathrm{rem}(x + y, p)$$
$$x \cdot y \bmod p = \mathrm{rem}(x \cdot y, p)$$

rem – remained (остаток от целочисл. деления)
Ex.: $p = 7, \mathbb{Z}_p = \{0, 1, 2, 3, 4, 5, 6\}$

$$5 + 6 \bmod p = \mathrm{rem}(11, 7) = 4$$
$$3 \cdot 3 \bmod p = \mathrm{rem}(9, 7) = 2$$

- For each non-zero $x \in \mathbb{Z}_p$, $\exists x^{-1} \in \mathbb{Z}_p$: $x \cdot x^{-1} = 1 \bmod p$
  Ex.: $1^{-1} = 1$, $2^{-1} = 4$, $3^{-1} = 5$, $4^{-1} = 2$, $5^{-1} = 3$, $6^{-1} = 6$ in $\mathbb{Z}_7$.
- The set of invertible elements is denoted $\mathbb{Z}_p^* = \mathbb{Z}_p \setminus \{0\}$

Math crash course I: structure of $\mathbb{Z}_p^*$

- Fermat's theorem: $g^{p-1} = 1 \bmod p \;\; \forall 0 \neq g \in \mathbb{Z}_p$
  Ex.: $2^6 = 64 = 1 \bmod 7$

- $\mathbb{Z}_p^*$ is a cyclic group, i.e.,
  $\exists g \in \mathbb{Z}_p$ s.t. $\mathbb{Z}_p^* = \{1 = g^0, g^1, g^2, \ldots, g^{p-2}\}$
  Ex. $\mathbb{Z}_7^* = \{1, 3, 3^2 = 2, 3^3 = 6, 3^4 = 4, 3^5 = 5\}$

Math crash course I: structure of $\mathbb{Z}_p^*$

- Fermat's theorem: $g^{p-1} = 1 \bmod p \ \ \forall 0 \neq g \in \mathbb{Z}_p$
  Ex.: $2^6 = 64 = 1 \bmod 7$
- $\mathbb{Z}_p^*$ is a cyclic group, i.e.,
  $\exists g \in \mathbb{Z}_p$ s.t. $\mathbb{Z}_p^* = \{1 = g^0, g^1, g^2, \ldots, g^{p-2}\}$
  Ex. $\mathbb{Z}_7^* = \{1, 3, 3^2 = 2, 3^3 = 6, 3^4 = 4, 3^5 = 5\}$
- Not every element is a generator of $\mathbb{Z}_p$, but for interesting fields we know how to do it efficiently

Math crash course I: structure of $\mathbb{Z}_p^*$

- Fermat's theorem: $g^{p-1} = 1 \bmod p \ \ \forall 0 \neq g \in \mathbb{Z}_p$
  Ex.: $2^6 = 64 = 1 \bmod 7$
- $\mathbb{Z}_p^*$ is a cyclic group, i.e.,
  $\exists g \in \mathbb{Z}_p$ s.t. $\mathbb{Z}_p^* = \{1 = g^0, g^1, g^2, \ldots, g^{p-2}\}$
  Ex. $\mathbb{Z}_7^* = \{1, 3, 3^2 = 2, 3^3 = 6, 3^4 = 4, 3^5 = 5\}$
- Not every element is a generator of $\mathbb{Z}_p$, but for interesting fields we know how to do it efficiently
- The order of $g \in \mathbb{Z}_p$, $\mathrm{ord}(g)$, is the smallest positive a s.t. $g^a = 1$
  For any generator g, $\mathrm{ord}(g) = p - 1$.

Math crash course I: computing in $\mathbb{Z}_p$

Arithmetic operations in $\mathbb{Z}_p$ are measured in $n = \lceil \log p \rceil$

- Addition is performed in $\mathcal{O}(n)$ bit operations
- Multiplication in $\mathcal{O}(n^2)$ (better in $\mathcal{O}(n^{1.7})$) bit operations
- Inversion in $\mathcal{O}(n^2)$ bit operations

Math crash course I: computing in $\mathbb{Z}_p$

Arithmetic operations in $\mathbb{Z}_p$ are measured in $n = \lceil \log p \rceil$

- Addition is performed in $\mathcal{O}(n)$ bit operations
- Multiplication in $\mathcal{O}(n^2)$ (better in $\mathcal{O}(n^{1.7})$) bit operations
- Inversion in $\mathcal{O}(n^2)$ bit operations
- Exponentiation x^r in time $\mathcal{O}(\log r)$ multiplications in $\mathbb{Z}_p$ (repeated squaring algorithm)
  - $y \leftarrow g$, $z \leftarrow 1$
  - **for** i **in** $[0, n]$**:**
  - **if** $r[i] == 1$ **:** $z \leftarrow z \cdot y$
  - $y \leftarrow y^2$
  - **return** z

Math crash course I: computing in $\mathbb{Z}_p$

Arithmetic operations in $\mathbb{Z}_p$ are measured in $n = \lceil \log p \rceil$

- Addition is performed in $\mathcal{O}(n)$ bit operations
- Multiplication in $\mathcal{O}(n^2)$ (better in $\mathcal{O}(n^{1.7})$) bit operations
- Inversion in $\mathcal{O}(n^2)$ bit operations
- Exponentiation x^r in time $\mathcal{O}(\log r)$ multiplications in $\mathbb{Z}_p$ (repeated squaring algorithm)
  - $y \leftarrow g$, $z \leftarrow 1$
  - **for** i **in** $[0, n]$**:**
  - **if** $r[i] == 1$ **:** $z \leftarrow z \cdot y$
  - $y \leftarrow y^2$
  - **return** z

  Ex.: compute g^r for $r = 23 = (10111)_2$. I.e., $g^{23} = g^{16+4+2+1}$.

$$g^1 \to g^{1+2} \to g^{1+2+4} \to g^{1+2+4} \to g^{1+2+4+16}$$

These operations are efficient in $\mathbb{Z}_p$.

Math crash course I: (believed to be) hard problems in $\mathbb{Z}_p$

1. The discrete logarithm problem (dlog):
   Given g – a generator of $\mathbb{Z}_p^*$ and $x \in \mathbb{Z}_p^*$, find r s.t. $g^r = x \bmod p$
   Example: Given $\langle 3 \rangle = \mathbb{Z}_7^*$ and 5 find $r = 5$ ($3^5 = 5$).

Math crash course I: (believed to be) hard problems in $\mathbb{Z}_p$

1. The discrete logarithm problem (dlog):
   Given g – a generator of $\mathbb{Z}_p^*$ and $x \in \mathbb{Z}_p^*$, find r s.t. $g^r = x \bmod p$
   Example: Given $\langle 3 \rangle = \mathbb{Z}_7^*$ and 5 find $r = 5$ ($3^5 = 5$).

2. The Computational Diffie-Hellman problem (CDH):
   Given g – a generator of $\mathbb{Z}_p^*$, $x = g^r \in \mathbb{Z}_p^*$, $y = g^t \in \mathbb{Z}_p^*$ find
   $z = g^{r \cdot t} \bmod p$.
   Example: Given $\langle 3 \rangle = \mathbb{Z}_7^*$ and $x = 2, y = 6$ find $z = 3^5 = 5 \bmod p$.

Math crash course I: (believed to be) hard problems in $\mathbb{Z}_p$

1. The discrete logarithm problem (dlog):
   Given g – a generator of $\mathbb{Z}_p^*$ and $x \in \mathbb{Z}_p^*$, find r s.t. $g^r = x \bmod p$
   Example: Given $\langle 3 \rangle = \mathbb{Z}_7^*$ and 5 find $r = 5$ ($3^5 = 5$).

2. The Computational Diffie-Hellman problem (CDH):
   Given g – a generator of $\mathbb{Z}_p^*$, $x = g^r \in \mathbb{Z}_p^*$, $y = g^t \in \mathbb{Z}_p^*$ find $z = g^{r \cdot t} \bmod p$.
   Example: Given $\langle 3 \rangle = \mathbb{Z}_7^*$ and $x = 2, y = 6$ find $z = 3^5 = 5 \bmod p$.

3. If one can solve dlog, one can also solve CDH

4. The other direction is not known in general

5. The hardness of both problems depend on the choice of p!
   Not every prime gives a hard instance of dlog/CDH.

NEVER CHOOSE p YOURSELF.

How hard are dlog/CDH?

Best known algorithm for computing dlog in $\mathbb{Z}_p^*$ for $n = \log p$:

General Number Field Sieve runs in time roughly $e^{n^{1/3}}$

This is sub-exponential runtime

How hard are dlog/CDH?

Best known algorithm for computing dlog in $\mathbb{Z}_p^*$ for $n = \log p$:

General Number Field Sieve runs in time roughly $e^{n^{1/3}}$

This is sub-exponential runtime
If we want to achieve the same level of security as a block-cipher with 128 bit key, we need

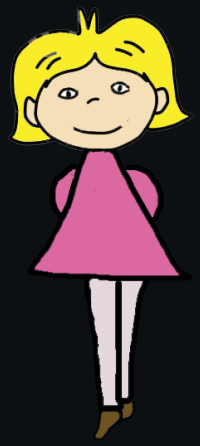
$$n \approx 3072 \text{ bits}$$

Instead of using $\mathbb{Z}_p^\star$, in practice we use Elliptic Curves to reduce the size of the group.

Diffie-Hellman Key Exchange

Fix a large prime p, and $\langle g \rangle = \mathbb{Z}_p^*$

Alice



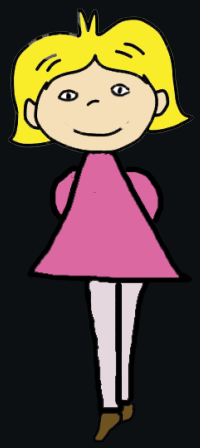
Bob

Diffie-Hellman Key Exchange

Fix a large prime p, and $\langle g \rangle = \mathbb{Z}_p^*$

Alice

$a \leftarrow \{2, \ldots, p-2\}$



Bob

$b \leftarrow \{2, \ldots, p-2\}$

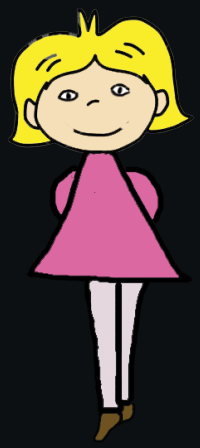
Diffie-Hellman Key Exchange

Fix a large prime p, and $\langle g \rangle = \mathbb{Z}_p^*$

Alice

$a \leftarrow \{2, \ldots, p-2\}$

$k_{\mathsf{A}} = g^a \bmod p$



$\xrightarrow{k_{\mathsf{A}}}$

Bob

$b \leftarrow \{2, \ldots, p-2\}$

$k_{\mathsf{B}} = g^b \bmod p$

Diffie-Hellman Key Exchange

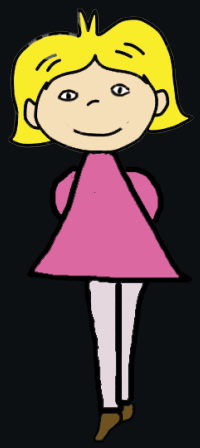
Fix a large prime p, and $\langle g \rangle = \mathbb{Z}_p^*$

Alice

Bob

$a \leftarrow \{2, \ldots, p-2\}$

$k_{\mathsf{A}} = g^a \bmod p$



k_{A} →

← k_{B}

$b \leftarrow \{2, \ldots, p-2\}$

$k_{\mathsf{B}} = g^b \bmod p$

Diffie-Hellman Key Exchange

Fix a large prime p, and $\langle g \rangle = \mathbb{Z}_p^*$

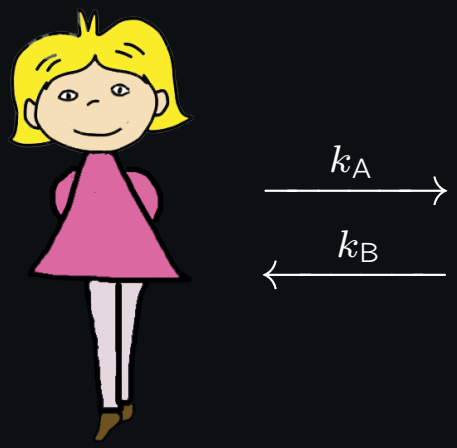
Alice

Bob

$a \leftarrow \{2, \ldots, p-2\}$

$k_{\mathsf{A}} = g^a \bmod p$

$k_{\mathsf{AB}} = k_{\mathsf{B}}^a$



$b \leftarrow \{2, \ldots, p-2\}$

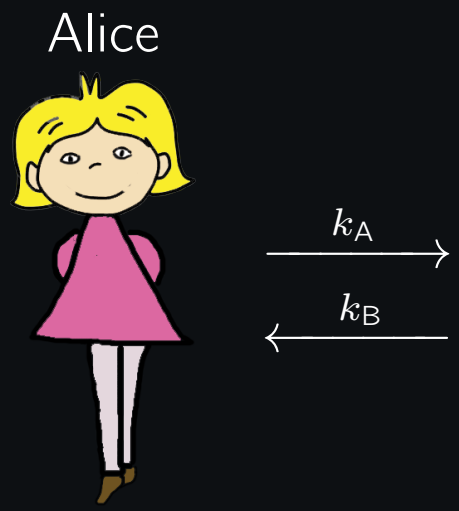
$k_{\mathsf{B}} = g^b \bmod p$

$k_{\mathsf{AB}} = k_{\mathsf{A}}^b$

k_{AB} – shared key

Diffie-Hellman Key Exchange

Fix a large prime p, and $\langle g \rangle = \mathbb{Z}_p^*$



k_{AB} – shared key

Correctness: $k_B^a = (g^b)^a = g^{ab} = (g^a)^b = k_A^b$.

Security(informally): adversary sees g^a, g^b. To get the shared key it should come up with g^{ab}.
This is an instance of the Computational Diffie-Hellman problem

Man-in-the-middle attack (active)

As described the protocol is insecure against active attacks.

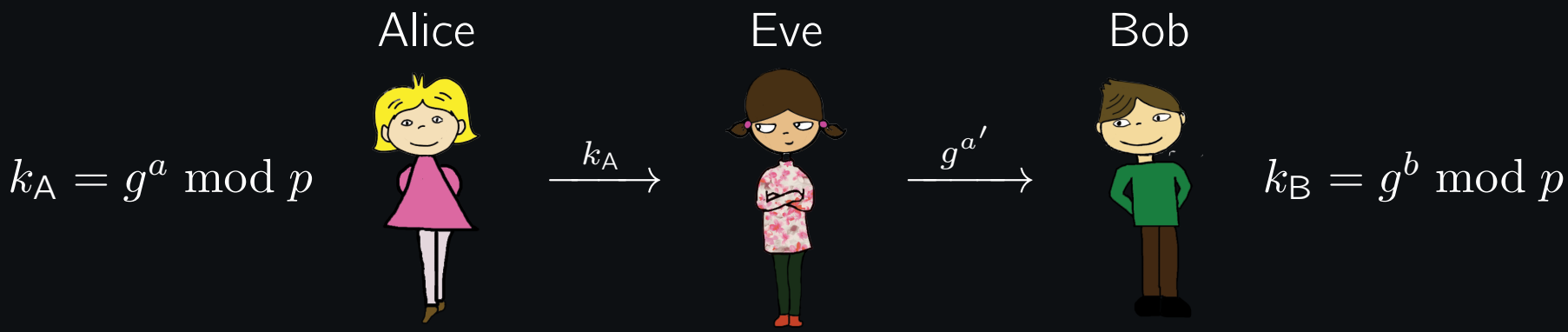
Alice

Eve

Bob

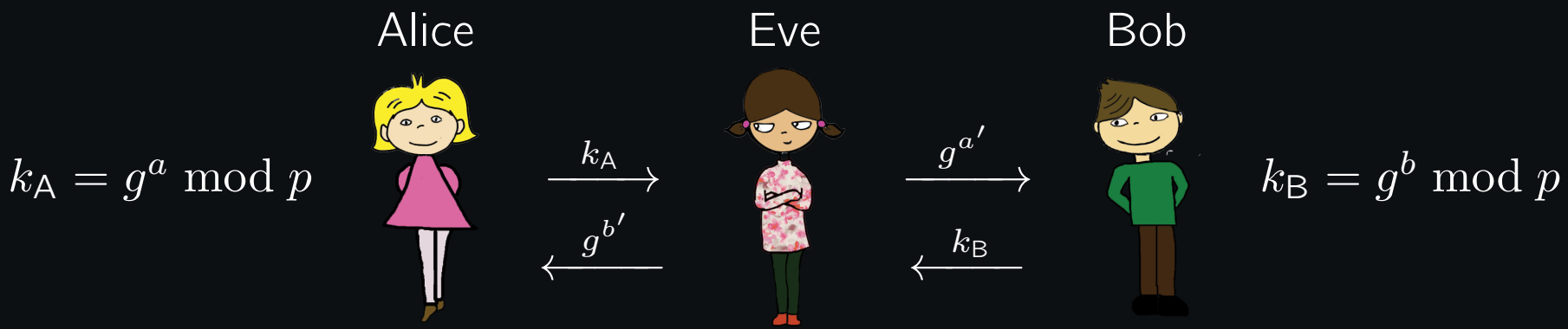
Man-in-the-middle attack (active)

As described the protocol is insecure against active attacks.



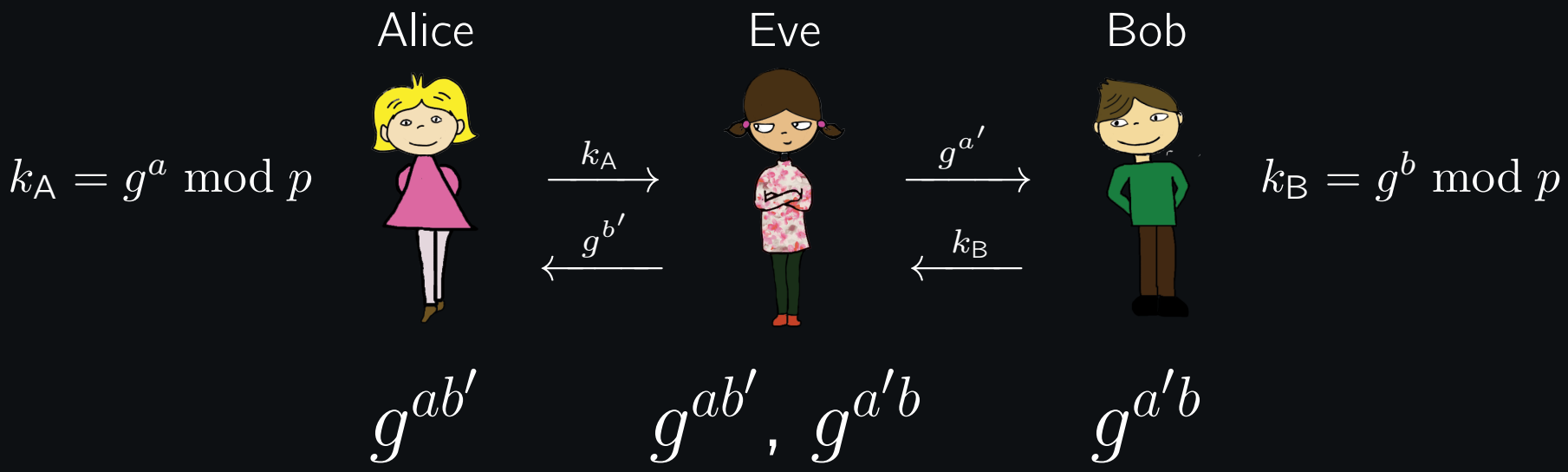
Man-in-the-middle attack (active)

As described the protocol is insecure against active attacks.



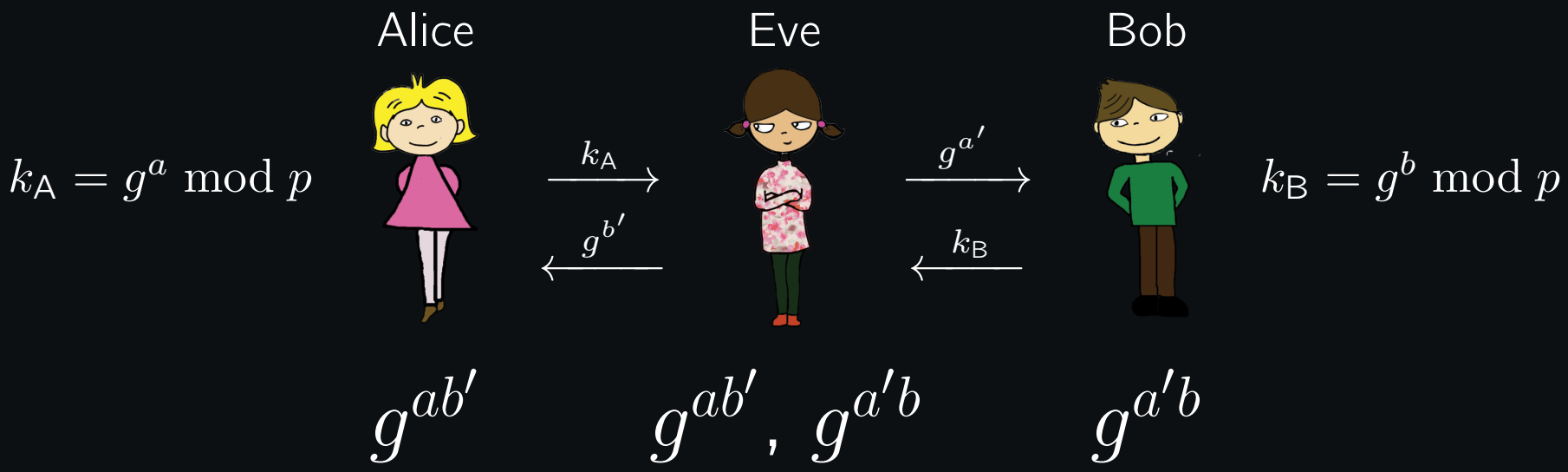
Man-in-the-middle attack (active)

As described the protocol is insecure against active attacks.



Man-in-the-middle attack (active)

As described the protocol is insecure against active attacks.



There are ways to fix this attack, you'll see them in the next lectures

Real world Diffie-Hellman

- The Man-in-the-Middle attack can be mitigated by using Authenticated Encryption (requires public key signature: wait until next lecture!)
- Real world Diffie-Hellman does not use in $\mathbb{Z}_p^*$ but Elliptic Curve, it's called ECDH
- It is non-trivial to build a secure group for the protocol. Don't try yourself, use standards
- For practical purposes we have a pool of parameters (Elliptic Curves), see e.g. `https://safecurves.cr.yp.to/`
- There is no GOST for Key Exchange, but it is being shipped with CryptoPRO software (there are recommended elliptic curves)

Programming Assignments 6

Task: implement Diffie-Hellman key exchange.

You may use OpenSSL in-built functions

You may chose any group available in OpenSSL
You can find the list of supported curves via

`openssl ecparam -list_curves`

To find the NID of the curve use
`https://github.com/openssl/openssl/blob/4e6647506331fc3b3ef5b23e5dbe188279ddd575/include/openssl/obj_mac.h`
There is no default curve!