
Lecture 10 – Diffie-Hellman key exchange III, computational aspects, Noise

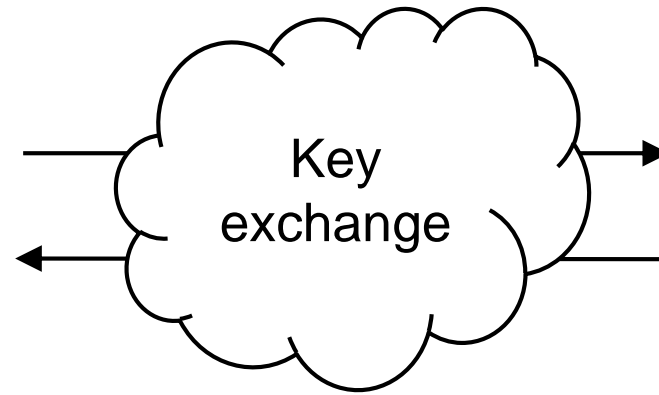
TEK4500

01.11.2023

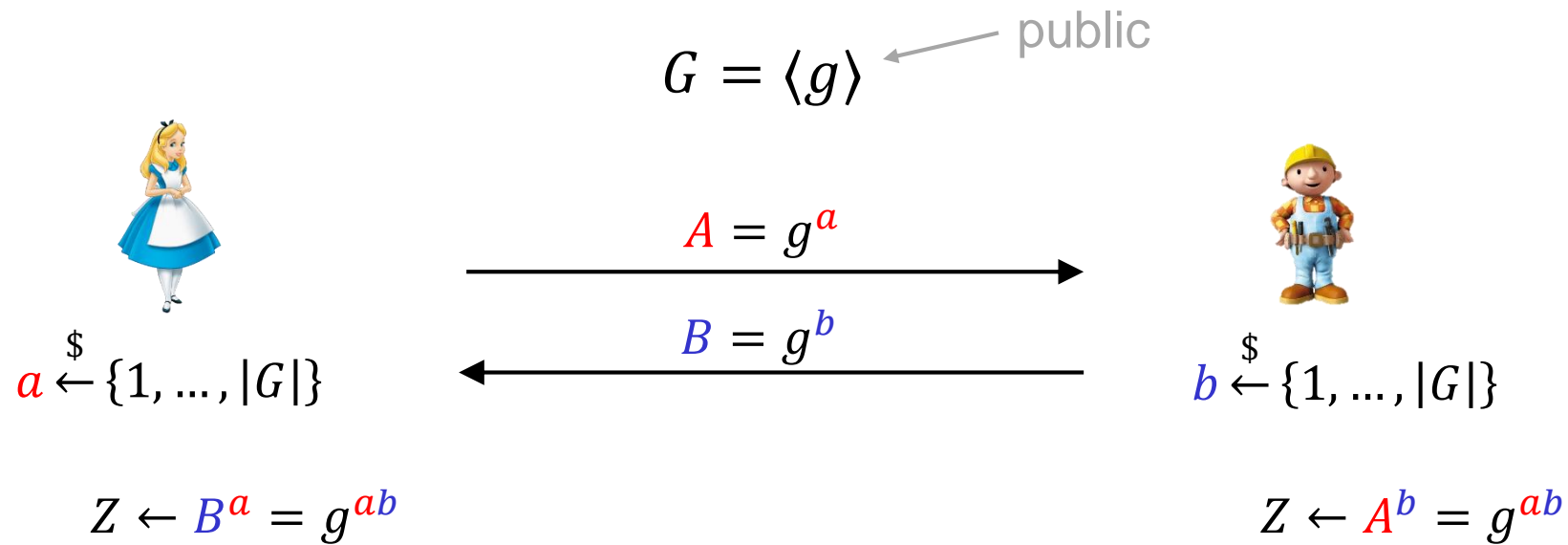
Håkon Jacobsen

hakon.jacobsen@its.uio.no

Diffie-Hellman



Diffie-Hellman – security



Examples

- $G = (\mathbf{Z}_p, +)$ not secure
- $G = (\mathbf{Z}_p^*, \cdot)$ secure for $|\mathbf{Z}_p^*|$ large enough (conjectured)
- $G = (E(\mathbf{F}_p), +)$ secure for $|E(\mathbf{F}_p)|$ large enough (conjectured)

Diffie-Hellman in (\mathbb{Z}_p^*, \cdot)

$p =$ 171254583176141379301960419792575778264088323240375085733932929816426671397476217788024387752387285929683446135893799323484756135034769321631669738132186983438164632891441853629126025225404949830905314972329658295365245072698488256583114202993359222957097432675083225259667739503949192575768420387716327420441424710535098501236058838158571626669177751934961573726561955583052720098912760065140004093658772181713883199238963093777917625906143118496429613802248519404604217104493688927252974870395873936387909672274883295377481008150475878590270591798350563488168080923804611822387520198054002990623911454389104774092183 $\approx 2^{2048}$



$A = 2$

$\text{mod } p$



323170060713110073003389139264238282488179412411402391
128420097514007417066343542226196894173635693471179017
379097041917546058732091950288537589861856221532121754
125149017745202702357960782362488842461894775876411059
286460994117232454266225221932305409190376805242355191
256797158701170010580558776510388618472802579760549035
697325615261670813393617995413364765591603683178967290
731783845896806396719009772021941686472258710314113364
293195361934716365332097170774482279885885653692086452
966360772502689555059283627511211740969729980684105543
595848665832916421362182310789909994486524682624169720
35911852507045361090559

$\$ \leftarrow \{1 \dots p\}$

665680077810100734070476416898417408020670352441378967
997224101900193957211854827569676933273935982997835638
067380762809024311842715496660113976613131478051784487
322446809608196031803691263486170912693625523249742383
264476433978409434004770395167011614217279552198029936
108023398643999746539201834933168220864789880837484536
563067997761270826642223539437541791058416731683176995
365899867720283590420932417377927106884773199080517477
615608675490982255098980545292032908358093913531433702
049429047413071525556307299541445514389721222256380833
743627991333762777307118317040885305789910094293548481
5501309561942770687524

$B = 2$

$\text{mod } p$

665680077810100734070476416898417408020670352441378967
997224101900193957211854827569676933273935982997835638
067380762809024311842715496660113976613131478051784487
322446809608196031803691263486170912693625523249742383
264476433978409434004770395167011614217279552198029936
108023398643999746539201834933168220864789880837484536
563067997761270826642223539437541791058416731683176995
365899867720283590420932417377927106884773199080517477
615608675490982255098980545292032908358093913531433702
049429047413071525556307299541445514389721222256380833
743627991333762777307118317040885305789910094293548481
5501309561942770687524

$\$ \leftarrow \{1 \dots p\}$

323170060713110073003389139264238282488179412411402391
128420097514007417066343542226196894173635693471179017
379097041917546058732091950288537589861856221532121754
125149017745202702357960782362488842461894775876411059
286460994117232454266225221932305409190376805242355191
256797158701170010580558776510388618472802579760549035
697325615261670813393617995413364765591603683178967290
731783845896806396719009772021941686472258710314113364
293195361934716365332097170774482279885885653692086452
966360772502689555059283627511211740969729980684105543
595848665832916421362182310789909994486524682624169720
35911852507045361090559

\times

665680077810100734070476416898417408020670352441378967
997224101900193957211854827569676933273935982997835638
067380762809024311842715496660113976613131478051784487
322446809608196031803691263486170912693625523249742383
264476433978409434004770395167011614217279552198029936
108023398643999746539201834933168220864789880837484536
563067997761270826642223539437541791058416731683176995
365899867720283590420932417377927106884773199080517477
615608675490982255098980545292032908358093913531433702
049429047413071525556307299541445514389721222256380833
743627991333762777307118317040885305789910094293548481
5501309561942770687524

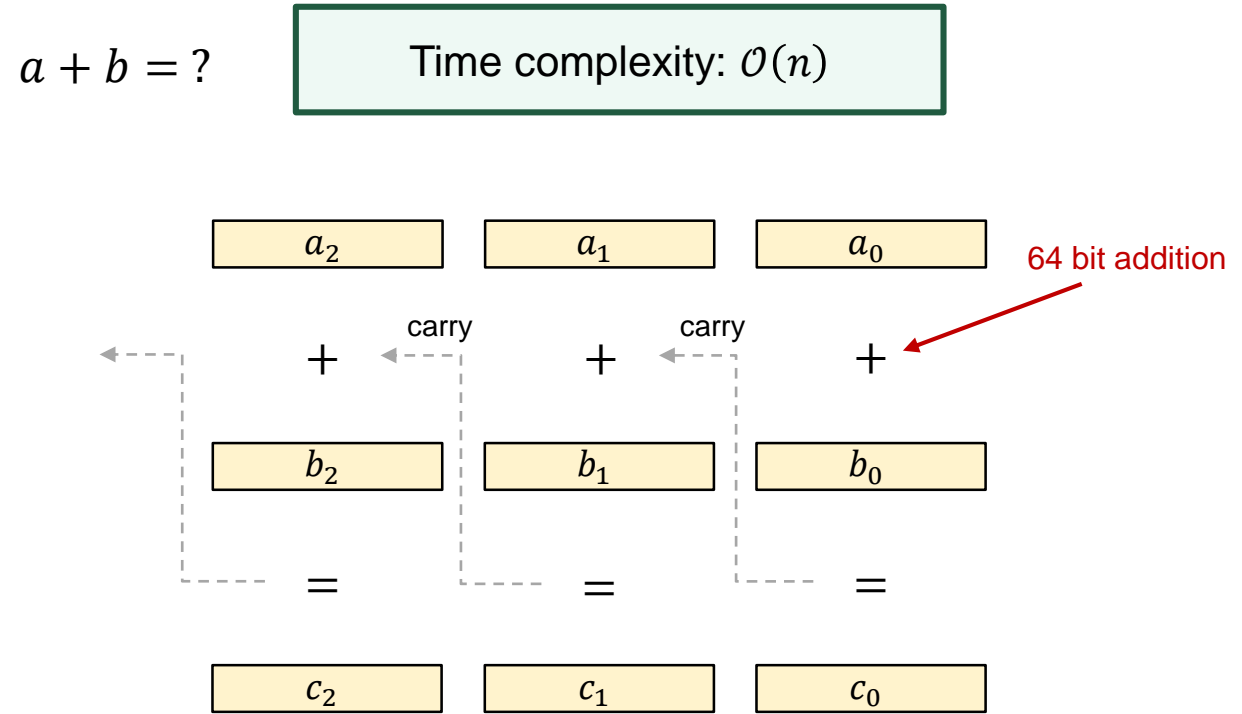
$Z \leftarrow 2$

$\text{mod } p$

Big-number arithmetic

$a =$ 3231700607131100730033891392642382824881794124114023911284200975140074170663435422261968941736356934711790173790970419175460587320919502885375898618562215321217541251490177452027023579607823624888424618947758764110592864609941172324542662252219323054091903768052423551912567971587011700105805587765103886184728025797605490356973256152616708133936179954133647655916036831789672907317838458968063967190097720219416864722587103141133642931953619347163653320971707744822798858856536920864529663607725026895505928362751121174096972998068410554359584866583291642136218231078990999448652468262416972035911852507045361090559

$b =$ 6656800778101007340704764168984174080206703524413789679972241019001939572118548275696769332739359829978356380673807628090243118427154966601139766131314780517844873224468096081960318036912634861709126936255232497423832644764339784094340047703951670116142172795521980299361080233986439997465392018349331682208647898808374845365630679977612708266422235394375417910584167316831769953658998677202835904209324173779271068847731990805174776156086754909822550989805452920329083580939135314337020494290474130715255563072995414455143897212222563808337436279913337627773071183170408853057899100942935484815501309561942770687524



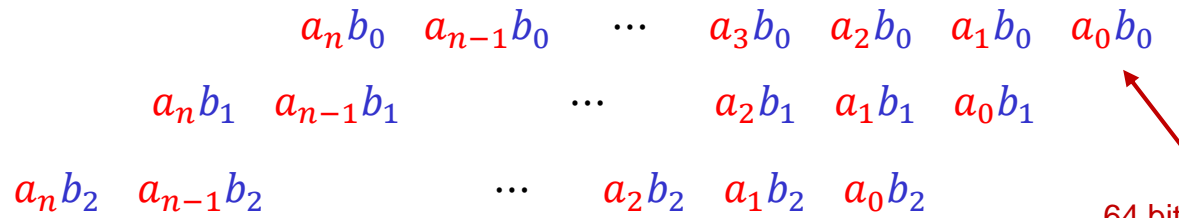
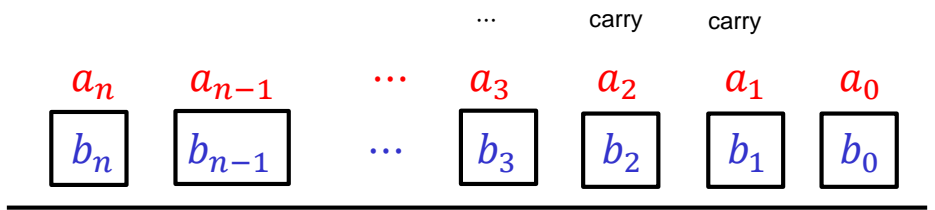
Big-number arithmetic

$a =$ 3231700607131100730033891392642382824881794124114023911284200975140074170663435422261968941736356934711790173790970419175460587320919502885375898618562215321217541251490177452027023579607823624888424618947758764110592864609941172324542662252219323054091903768052423551912567971587011700105805587765103886184728025797605490356973256152616708133936179954133647655916036831789672907317838458968063967190097720219416864722587103141133642931953619347163653320971707744822798858856536920864529663607725026895505928362751121174096972998068410554359584866583291642136218231078990999448652468262416972035911852507045361090559

$b =$ 6656800778101007340704764168984174080206703524413789679972241019001939572118548275696769332739359829978356380673807628090243118427154966601139766131314780517844873224468096081960318036912634861709126936255232497423832644764339784094340047703951670116142172795521980299361080233986439997465392018349331682208647898808374845365630679977612708266422235394375417910584167316831769953658998677202835904209324173779271068847731990805174776156086754909822550989805452920329083580939135314337020494290474130715255563072995414455143897212222563808337436279913337627773071183170408853057899100942935484815501309561942770687524

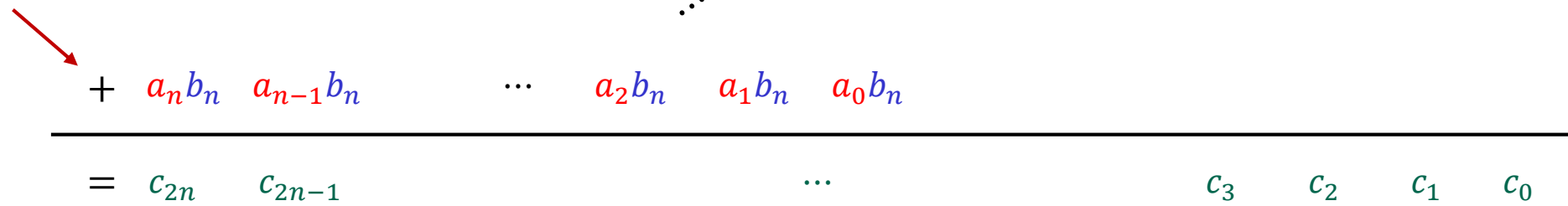
Time complexity: $\mathcal{O}(n^2)$

$a \cdot b = ?$

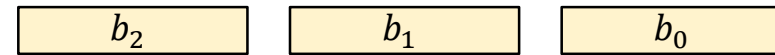
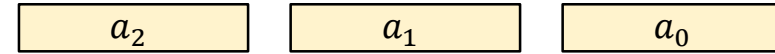
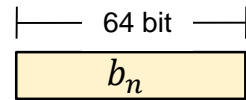
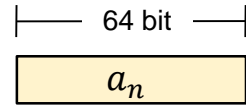


64 bit multiplication

64 bit addition



Big-number arithmetic



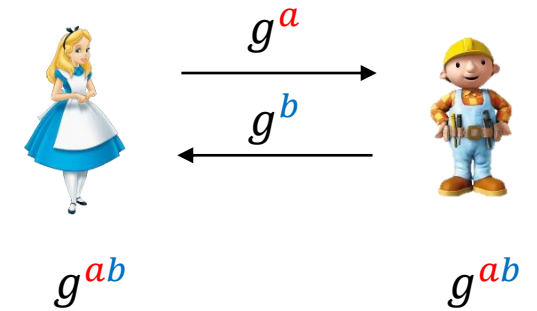
Algorithm	Input	Output	Time
ADD	a, b	$a + b$	$\mathcal{O}(n)$
MULT	a, b	ab	$\mathcal{O}(n^2)$
MOD	a, p	$a \bmod p$	$\mathcal{O}(n^2)$
MOD-INV	a	$a^{-1} \bmod p$	$\mathcal{O}(n^2)$

$\mathcal{O}(n \log n)^*$

* $n > 2^{713739807325663489766475852620783120641}$

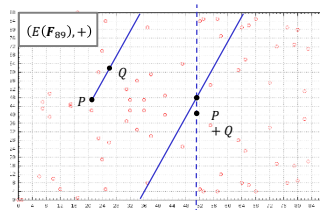
Diffie-Hellman – computations

- (\mathbf{Z}_p^*, \cdot)
 - Find prime p (one-time)
 - Find generator $\langle g \rangle = (\mathbf{Z}_p^*, \cdot)$ (one-time)
 - Compute $g^a \bmod p = \underbrace{gg \cdots g}_{a \text{ times}} \bmod p$ (every DH exchange)
 - Multiply $x \cdot y \bmod p$



- $(E(\mathbf{F}_p), +)$
 - Find prime p (one-time)
 - Generate curve $y^2 = x^3 + ax + b \bmod p$ (one-time)
 - Find curve group order $|E(\mathbf{F}_p)|$ (one-time)
 - Find generator Q (one-time)
 - Compute $aQ = \underbrace{Q + Q + \cdots + Q}_a$ (every DH exchange)
 - Point addition

Group exponentiation g^a

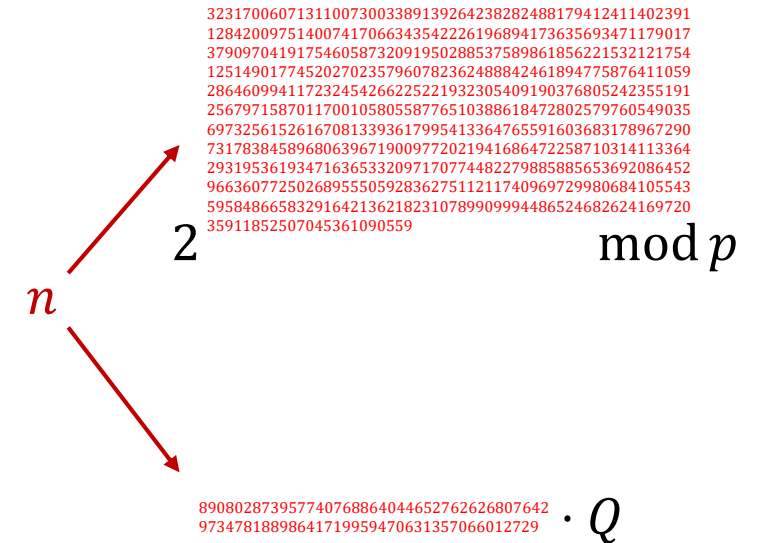


Computing in groups – exponentiation

Goal: compute $g^n \in G$

Naïve approach: $g^n = \overbrace{g \circ g \circ g \circ \dots \circ g}^n$

- Running time: n group operations
- (\mathbf{Z}_p^*, \cdot) : $n \approx 2^{2048}$
- $(E(\mathbf{F}_p), +)$: $n \approx 2^{256}$
- Not feasible



Computing in groups – exponentiation; square-and-multiply

Goal: compute $g^n \in G$

$$n = (b_5 b_4 b_3 b_2 b_1 b_0)_2$$

$$= b_5 \cdot 2^5 + b_4 \cdot 2^4 + b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0$$

$$y \leftarrow e$$

$$y \leftarrow y^2 \circ g^{b_5} = g^{b_5}$$

$$y \leftarrow y^2 \circ g^{b_4} = g^{2b_5 + b_4}$$

$$y \leftarrow y^2 \circ g^{b_3} = g^{2^2 b_5 + 2b_4 + b_3}$$

$$y \leftarrow y^2 \circ g^{b_2} = g^{2^3 b_5 + 2^2 b_4 + 2b_3 + b_2}$$

$$y \leftarrow y^2 \circ g^{b_1} = g^{2^4 b_5 + 2^3 b_4 + 2^2 b_3 + 2b_2 + b_1}$$

$$y \leftarrow y^2 \circ g^{b_0} = g^{2^5 b_5 + 2^4 b_4 + 2^3 b_3 + 2^2 b_2 + 2b_1 + b_0}$$

$$g^{43}$$

$$43 = 101011_2$$

$$y \leftarrow e$$

$$y \leftarrow y^2 \circ g^1 = g^1$$

$$y \leftarrow y^2 \circ g^0 = g^2$$

$$y \leftarrow y^2 \circ g^1 = g^{4+1} = g^5$$

$$y \leftarrow y^2 \circ g^0 = g^{10}$$

$$y \leftarrow y^2 \circ g^1 = g^{20+1} = g^{21}$$

$$y \leftarrow y^2 \circ g^1 = g^{42+1} = \underline{g^{43}}$$

Computing in groups – exponentiation; square-and-multiply

Goal: compute $g^n \in G$

g^{43}

$$n = (b_5 b_4 b_3 b_2 b_1 b_0)_2 \\ = b_5 \cdot 2^5 + b_4 \cdot 2^4 + b_3 \cdot 2^3 + b_2 \cdot 2^2 + b_1 \cdot 2^1 + b_0$$

$$43 = 101011_2$$

$$y \leftarrow e$$

$$y \leftarrow y^2 \circ g^{b_5} = g^{b_5}$$

$$y \leftarrow y^2 \circ g^{b_4} = g^{2b_5 + b_4}$$

$$y \leftarrow y^2 \circ g^{b_3} = g^{2^2 b_5 + 2b_4 + b_3}$$

$$y \leftarrow y^2 \circ g^{b_2} = g^{2^3 b_5 + 2^2 b_4 + 2b_3 + b_2}$$

$$y \leftarrow y^2 \circ g^{b_1} = g^{2^4 b_5 + 2^3 b_4 + 2^2 b_3 + 2b_2 + b_1}$$

$$y \leftarrow y^2 \circ g^{b_0} = g^{2^5 b_5 + 2^4 b_4 + 2^3 b_3 + 2^2 b_2 + 2b_1 + b_0}$$

Square&Multiply($g \in G, n \in \mathbb{Z}$)

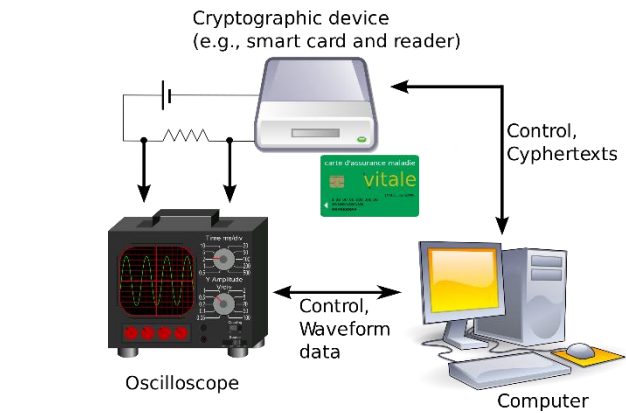
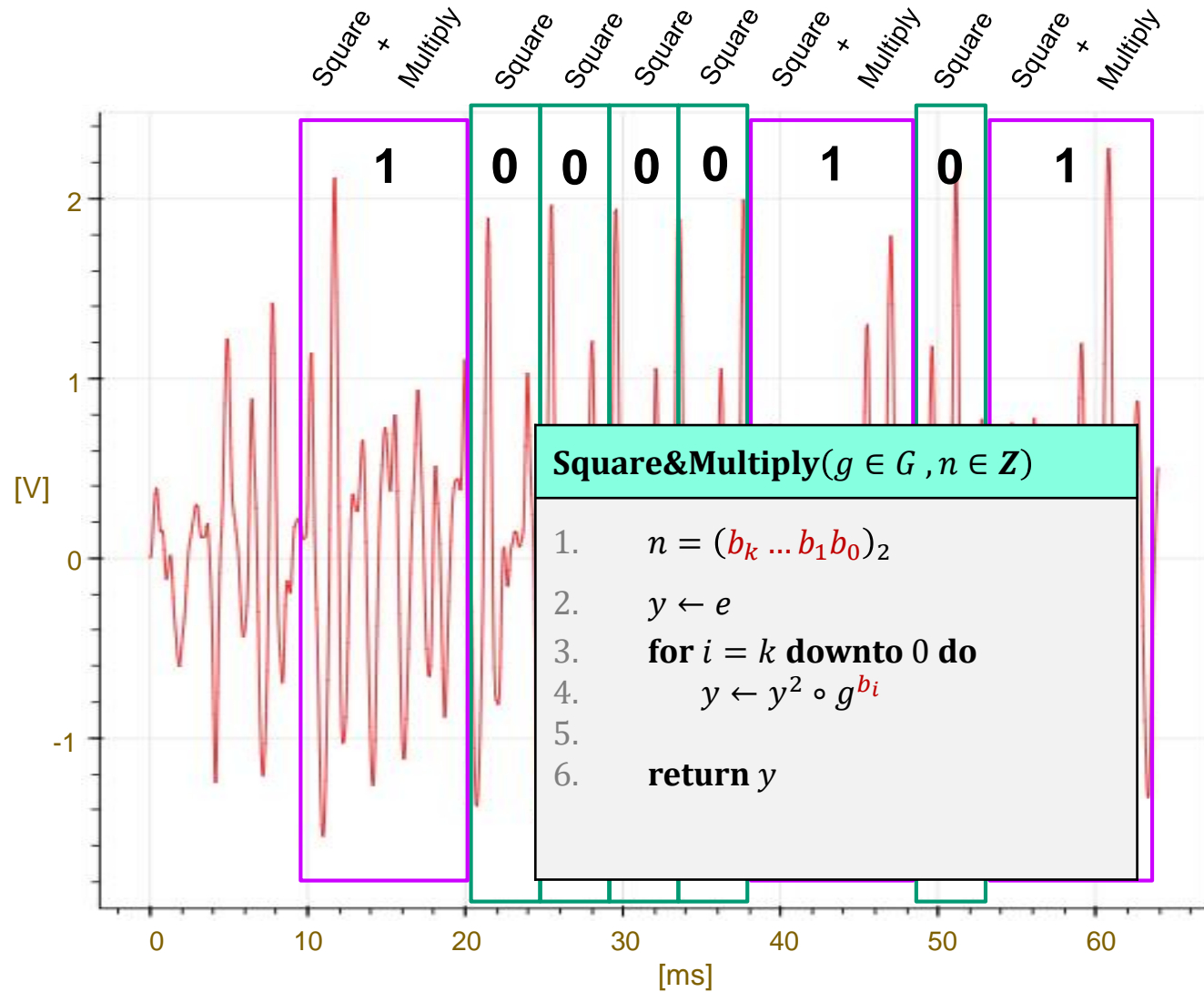
1. $n = (b_k \dots b_1 b_0)_2$
2. $y \leftarrow e$
3. **for** $i = k$ **downto** 0 **do**
4. $y \leftarrow y^2 \circ g^{b_i}$
- 5.
6. **return** y

Square-and-multiply

- (\mathbf{Z}_p^*, \cdot)
 - Naïve exponentiation: $\mathcal{O}(n) \approx p$ $\approx 2^{2048}$
 - Square-and-multiply: $\mathcal{O}(\log_2 n) \approx \log_2 p$ ≈ 2048

- $(E(\mathbf{F}_p), +)$
 - Naïve "exponentiation" (i.e., point multiplication nQ): $\mathcal{O}(n) \approx |E(\mathbf{F}_p)|$ $\approx 2^{256}$
 - "Square-and-multiply" (i.e., double-and-add): $\mathcal{O}(\log_2 n) \approx \log_2 |E(\mathbf{F}_p)|$ ≈ 256

Side-channel attacks – power analysis



Square&Multiply($g \in G, n \in \mathbb{Z}$)

1. $n = (b_k \dots b_1 b_0)_2$
2. $y \leftarrow e$
3. **for** $i = k$ **downto** 0 **do**
4. $y \leftarrow y^2$
5. **if** $b_i = 1$ **then**
6. $y \leftarrow y \circ g$
7. **return** y

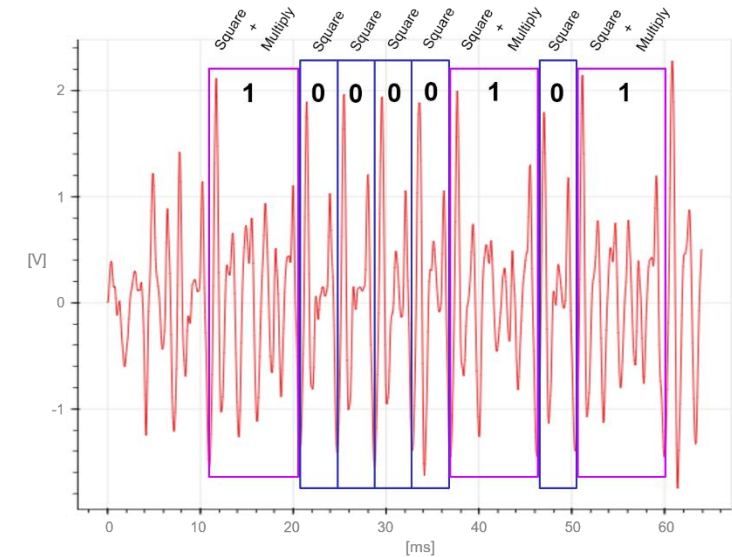
Mitigations

- Power (and time) difference between $b_i = 0$ and $b_i = 1$
- Solution: Square-and-Always-Multiply

```

Square-and-always-Multiply( $g \in G, n \in \mathbb{Z}$ )
1.   $n = (b_k \dots b_1 b_0)_2$ 
2.   $y \leftarrow e$ 
3.  for  $i = k$  downto 0 do
4.       $y \leftarrow y^2$ 
5.       $z \leftarrow y \circ g$ 
6.      if  $b_i = 1$  then
7.           $y \leftarrow z$ 
8.  return  $y$ 
    
```

- Wastes computations: $\approx t/2$ of multiplications thrown away
- Better solutions exist



```

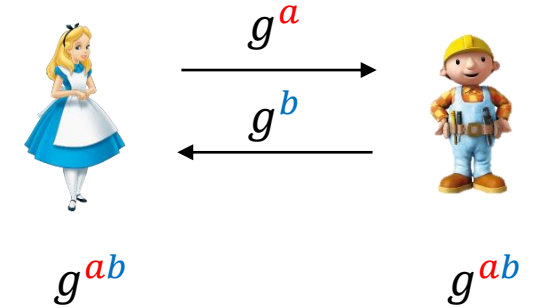
Square&Multiply( $g \in G, n \in \mathbb{Z}$ )
1.   $n = (b_k \dots b_1 b_0)_2$ 
2.   $y \leftarrow 1$ 
3.  for  $i = k$  downto 0 do
4.       $y \leftarrow y^2$ 
5.      if  $b_i = 1$  then
6.           $y \leftarrow y \circ g$ 
7.  return  $y$ 
    
```

Diffie-Hellman – computations

- (\mathbf{Z}_p^*, \cdot)

- Find prime p
- Find generator $\langle g \rangle = (\mathbf{Z}_p^*, \cdot)$
- Compute $g^a = gg \cdots g \pmod p$
 - Multiply $x \cdot y \pmod p$

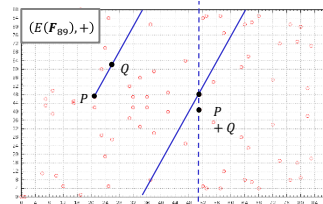
(one-time)
 (one-time)
 (every DH exchange)



- $(E(\mathbf{F}_p), +)$

- Find prime p
- Generate curve $y^2 = x^3 + ax + b \pmod p$
- Find curve group order $|E(\mathbf{F}_p)|$
- Find generator Q
- Compute $aQ = Q + Q + \cdots + Q$
 - Point addition

(one-time)
 (one-time)
 (one-time)
 (one-time)
 (every DH exchange)



Finding large prime numbers

- We need *large* prime numbers for both (\mathbf{Z}_p^*, \cdot) and $(E(\mathbf{F}_p), +)$
- No efficient "prime-generating" formula is known
- Simple idea: pick random number and check if it's prime
- Efficient? \Rightarrow What is the success probability?
 - Depends on the ratio primes / non-primes
 - $\pi(n) \stackrel{\text{def}}{=} \text{"\#prime numbers } \leq n\text{"}$
 - **Prime Number Theorem:** $\frac{\pi(n)}{n} \approx \frac{1}{\ln n}$
 - 2048 bit number: ≈ 700 trials needed

```

GenPrime(k)
1. while true do
2.    $n \stackrel{\$}{\leftarrow}$  odd  $k$  bit integer
3.   if IsPrime(n) then
4.     return n
    
```

$$n \approx 2^{2048} \Rightarrow \frac{\pi(n)}{n} \approx \frac{1}{\ln 2^{2048}} \approx \frac{1}{1400}$$

n	100	1000	10^6	10^9	10^{12}	10^{15}
$\pi(n)$	25	168	$\approx 10^3$	$\approx 10^6$	$\approx 10^9$	$\approx 10^{12}$

Finding large prime numbers

- GenPrime efficient?

- prime-to-non-prime ratio ✓
- IsPrime(n) ?

- Naïve IsPrime(n):

- 3 divides n ? Not prime
- 5 divides n ? Not prime
- 7 divides n ? Not prime
- \vdots
- $\lfloor \sqrt{n} \rfloor$ divides n ? Not prime
- n is prime!

- Very inefficient: $n \approx 2^{2048} \Rightarrow \sqrt{\pi(n)} \approx \sqrt{\frac{2^{2048}}{\ln 2^{2048}}} \approx 2^{1000}$

- Want: a simple property which *only* holds for prime numbers

```
GenPrime( $k$ )  
1. while true do  
2.    $n \xleftarrow{\$}$  odd  $k$ -bit integer  
3.   if IsPrime( $n$ ) then  
4.     return  $n$ 
```

Fermat's theorem

Theorem: if (G, \circ) is a finite group, then for all $g \in G$:

$$g^{|G|} = e$$

$$(\mathbf{Z}_p^*, \cdot) = \{1, 2, \dots, p-1\}$$

$$|\mathbf{Z}_p^*| = p-1$$

Fermat's theorem: if p is prime, then for all $a \not\equiv 0 \pmod{p}$:

$$a^{p-1} \equiv 1 \pmod{p}$$

A

B

p is prime \Rightarrow all $a \not\equiv 0 \pmod{p}$ satisfy Fermat's theorem

p is **not** prime \Leftarrow some $a \not\equiv 0 \pmod{p}$ does **not** satisfy Fermat's theorem

\bar{A}

\bar{B}

Fermat primality test

Fermat's theorem: if p is prime, then for all $a \neq 0 \pmod{p}$:

$$a^{p-1} \equiv 1 \pmod{p}$$

$$2^{560} = 1 \pmod{561}$$

$$3^{560} = 1 \pmod{561}$$

$$4^{560} = 1 \pmod{561}$$

⋮

$$560^{560} = 1 \pmod{561}$$

$$561 = 3 \cdot 187$$

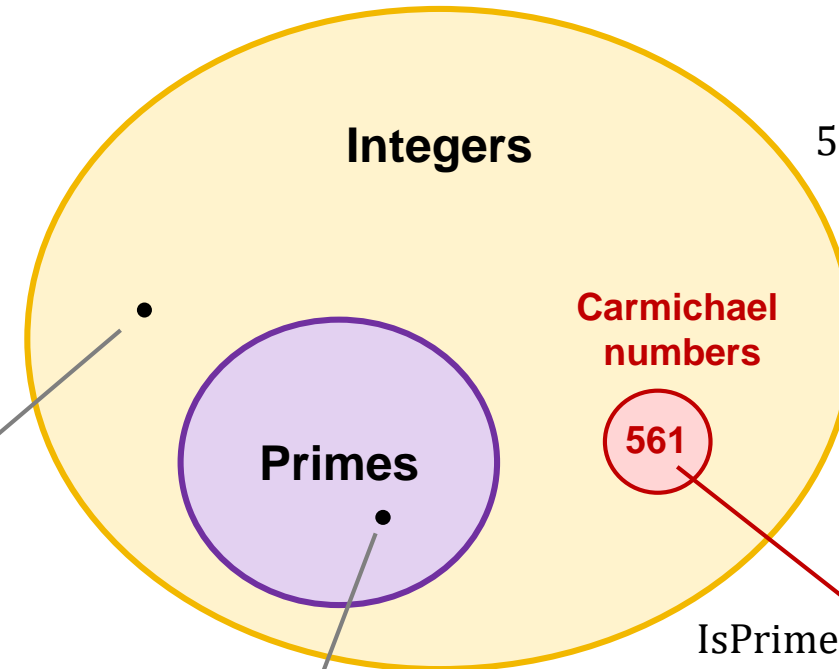
```

IsPrime(n)
1. for  $a \in \{2, 3, \dots, t\}$  do
2.   if  $a^{n-1} \not\equiv 1 \pmod{n}$  then
3.     return false
4. return true
    
```

IsPrime(n) = false

IsPrime(n) = true

IsPrime(n) = true



Better tests exist:

- Euler-test (Solovay-Strassen algorithm)
- Strong pseudoprime-test (Miller-Rabin algorithm)

Finding primes

IsPrime(n) // Fermat

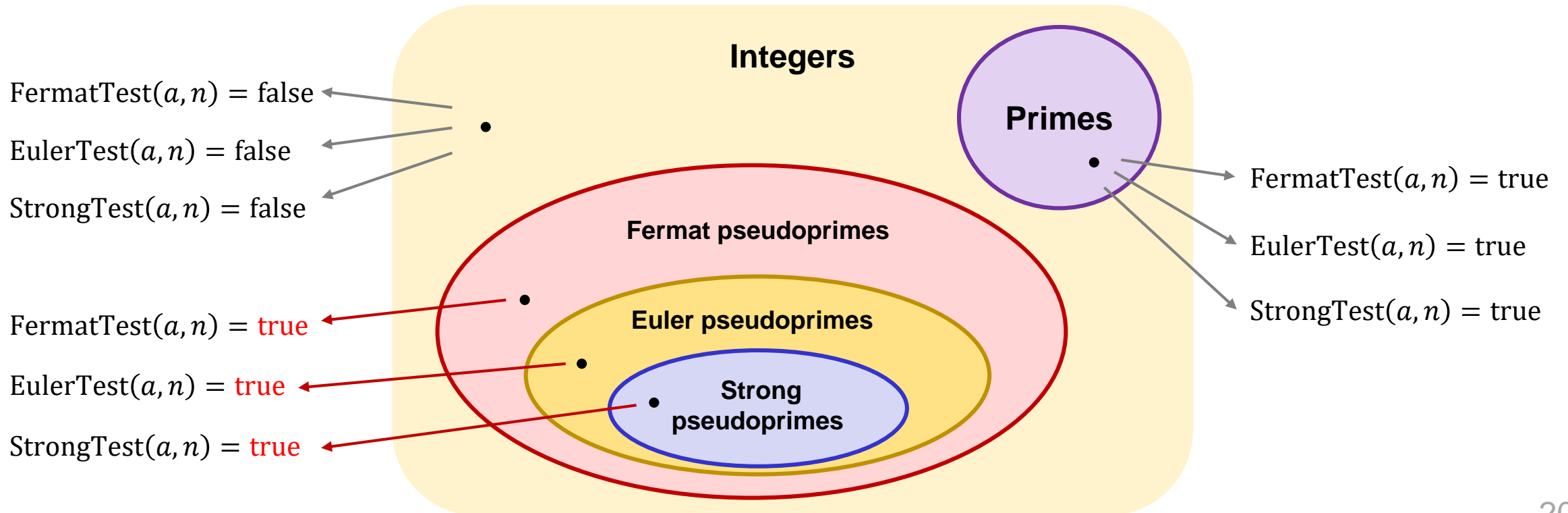
1. for $a \in \{2, 3, \dots, t\}$ do
2. if $\neg \text{FermatTest}(a, n)$ then
3. return false
4. return true

IsPrime(n) // Solovay-Strassen

1. for $a \in \{2, 3, \dots, t\}$ do
2. if $\neg \text{EulerTest}(a, n)$ then
3. return false
4. return true

IsPrime(n) // Miller-Rabin

1. for $a \in \{2, 3, \dots, t\}$ do
2. if $\neg \text{StrongTest}(a, n)$ then
3. return false
4. return true



Finding primes in practice

- Miller-Rabin most used in practice
- Failure probability $\leq \frac{1}{4^t}$
- What about *deterministic* primality tests?
 - Open problem for a long time
 - *Agrawal, Kayal & Saxena '02*: "PRIMES is in P"
 - Original running time $\approx O(k^{12})$
 - Quickly improved to $\approx O(k^6)$

```
IsPrime(n) // Miller-Rabin
```

```
1. for a ∈ {2, 3, ..., t} do
2.     if ¬StrongTest(a, n) then
3.         return false
4. return true
```

Annals of Mathematics, 160 (2004), 781–793

PRIMES is in P

By MANINDRA AGRAWAL, NEERAJ KAYAL, and NITIN SAXENA*

Abstract

We present an unconditional deterministic polynomial-time algorithm that determines whether an input number is prime or composite.

1. Introduction

Prime numbers are of fundamental importance in mathematics in general, and number theory in particular. So it is of great interest to study different properties of prime numbers. Of special interest are those properties that allow one to determine efficiently if a number is prime. Such efficient tests are also useful in practice: a number of cryptographic protocols need large prime numbers.

Finding generators

- Easy in prime-order groups
 - All non-identity elements are generators!
- (\mathbb{Z}_p^*, \cdot)
 - Not prime-order!
 - Common in practice: pick element at random; check if generator

Lagrange's theorem:

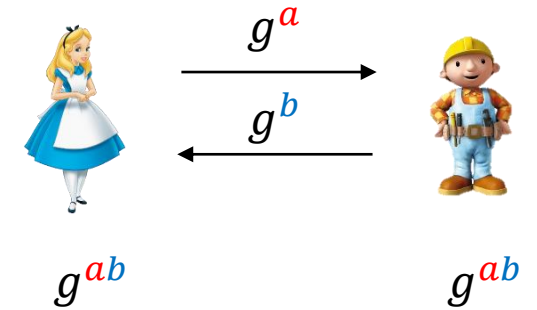
if $H < G$ then $|H|$ divides $|G|$

Diffie-Hellman – computations

- (\mathbf{Z}_p^*, \cdot)

- Find prime p
- Find generator $\langle g \rangle = (\mathbf{Z}_p^*, \cdot)$
- Compute $g^a = gg \cdots g \pmod p$
 - Multiply $x \cdot y \pmod p$

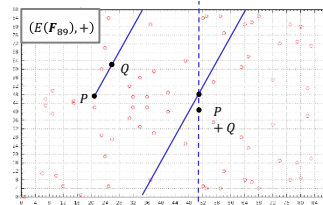
(one-time)
 (one-time)
 (every DH exchange)



- $(E(\mathbf{F}_p), +)$

- Find prime p
- Generate curve $y^2 = x^3 + ax + b \pmod p$
- Find curve group order $|E(\mathbf{F}_p)|$
- Find generator Q
- Compute $aQ = Q + Q + \cdots + Q$
 - Point addition

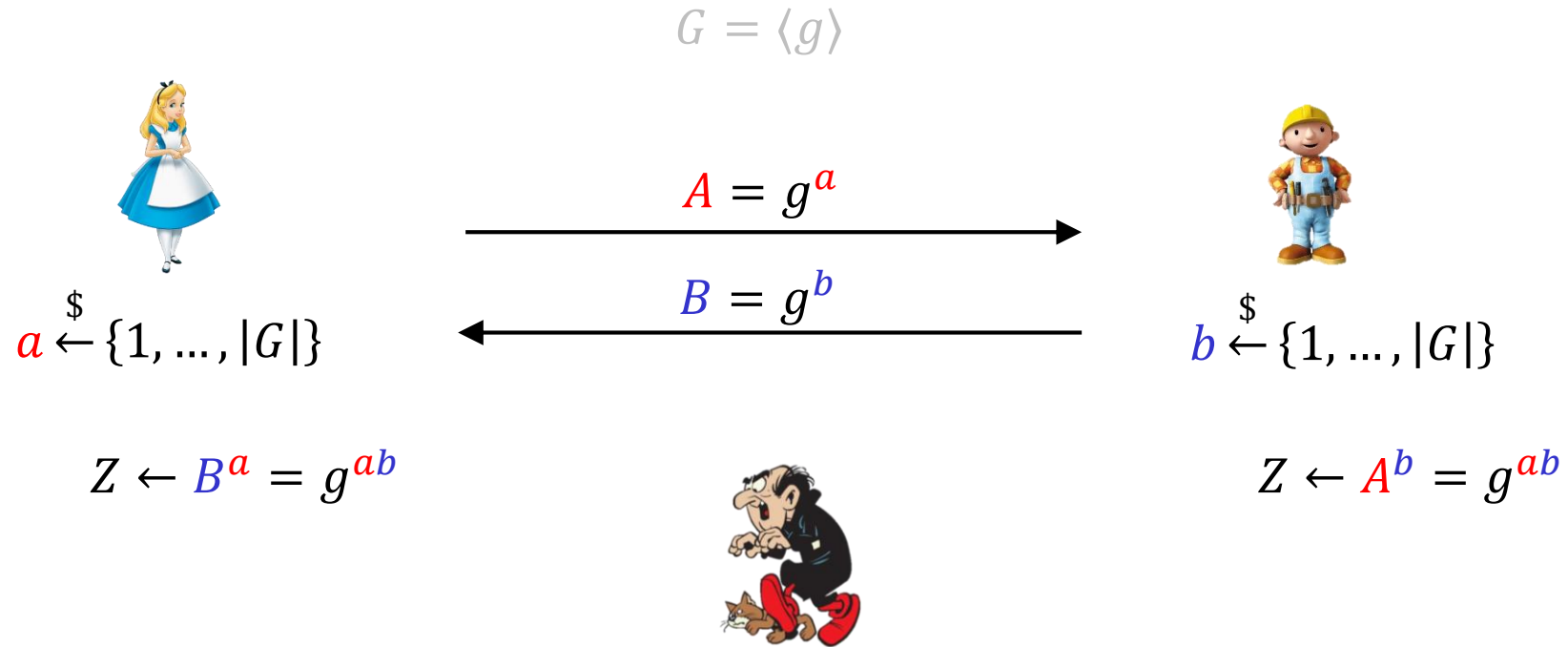
(one-time)
 (one-time) Tricky! Standardized choices (NIST P-256, Curve25519)
 (one-time) Tricky! (Schoof's algorithm) $|E(\mathbf{F}_p)| \neq p$ (typically)
 (one-time) Easy! $|E(\mathbf{F}_p)|$ usually prime
 (every DH exchange) (or close to prime)



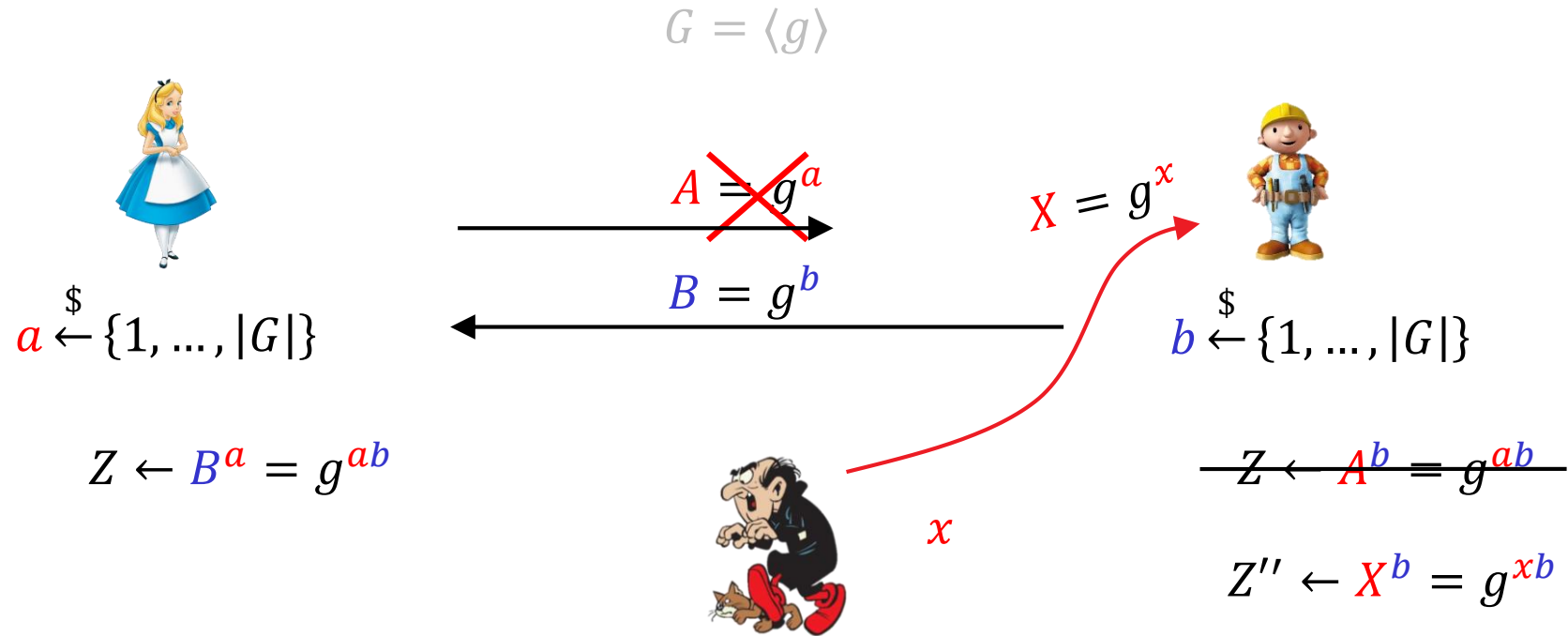


Attacking Diffie-Hellman

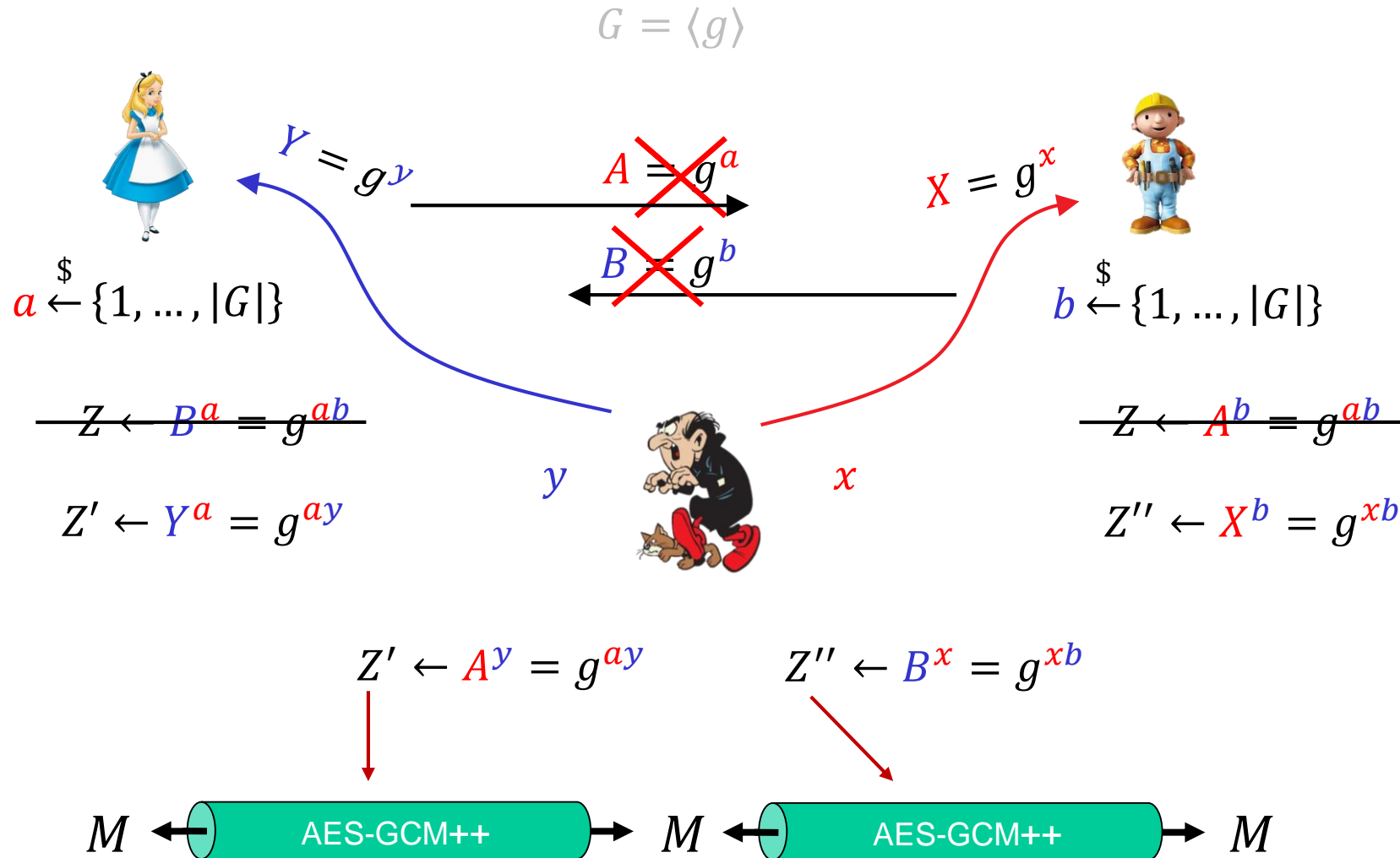
Diffie-Hellman – man-in-the-middle attack



Diffie-Hellman – man-in-the-middle attack



Diffie-Hellman – man-in-the-middle attack



Noise-protocol

Long-term key $\rightarrow A = g^a$



$x \xleftarrow{\$} \{1, \dots, |G|\}$

$G = \langle g \rangle, A, B$

$B = g^b \leftarrow$ Long-term key



$y \xleftarrow{\$} \{1, \dots, |G|\}$

$X = g^x$

$Y = g^y$

$$\begin{aligned} K &\leftarrow H(X, Y, B^a, Y^x) \\ &= H(X, Y, g^{ab}, g^{xy}) \end{aligned}$$

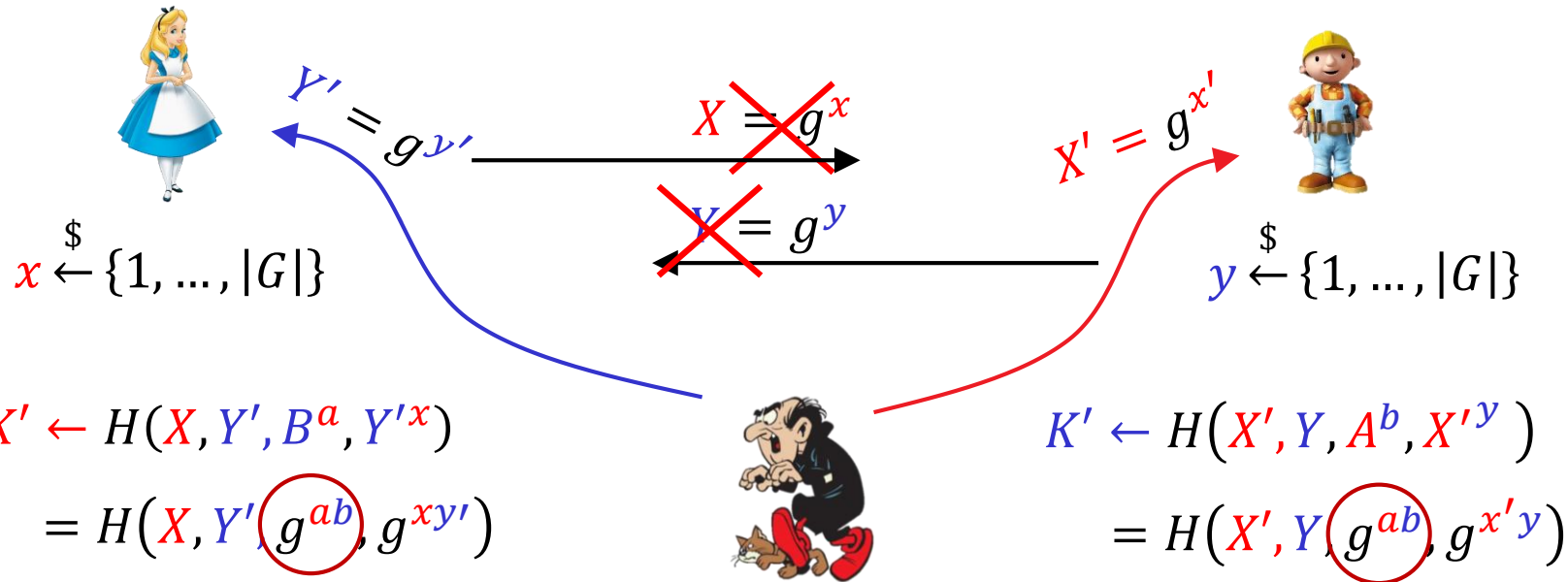
$$\begin{aligned} K &\leftarrow H(X, Y, A^b, X^y) \\ &= H(X, Y, g^{ab}, g^{xy}) \end{aligned}$$

Noise-protocol

Long-term key $\rightarrow A = g^a$

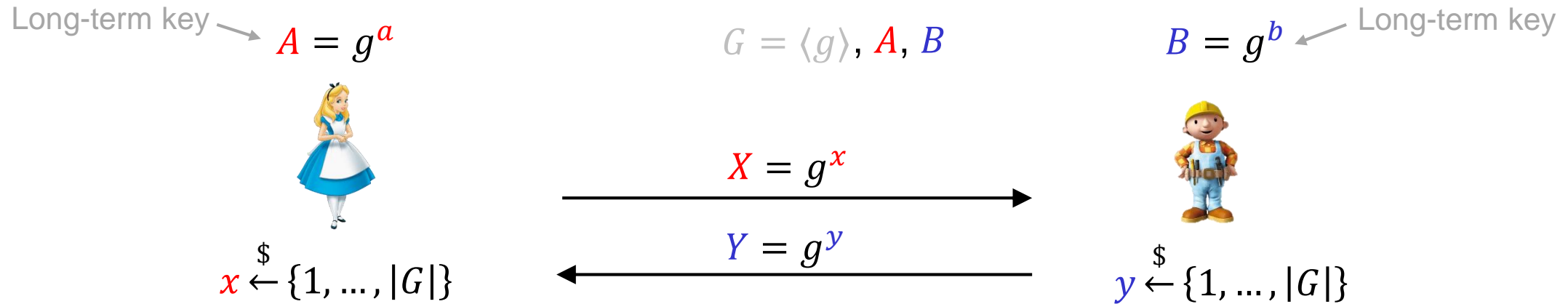
$G = \langle g \rangle, A, B$

$B = g^b$ \leftarrow Long-term key



Can't compute K' or K' !

Noise-protocol



Many alternatives:

$$K \leftarrow H(X, Y, g^{ab}, g^{xy})$$

$$K \leftarrow H(X, Y, g^{ay}, g^{xb})$$

$$K \leftarrow H(X, Y, g^{ay}, g^{xb}, g^{xy})$$

$$K \leftarrow H(X, Y, g^{ab}, g^{ay}, g^{xb}, g^{xy})$$

static-static, ephemeral-ephemeral

static-ephemeral, ephemeral-static

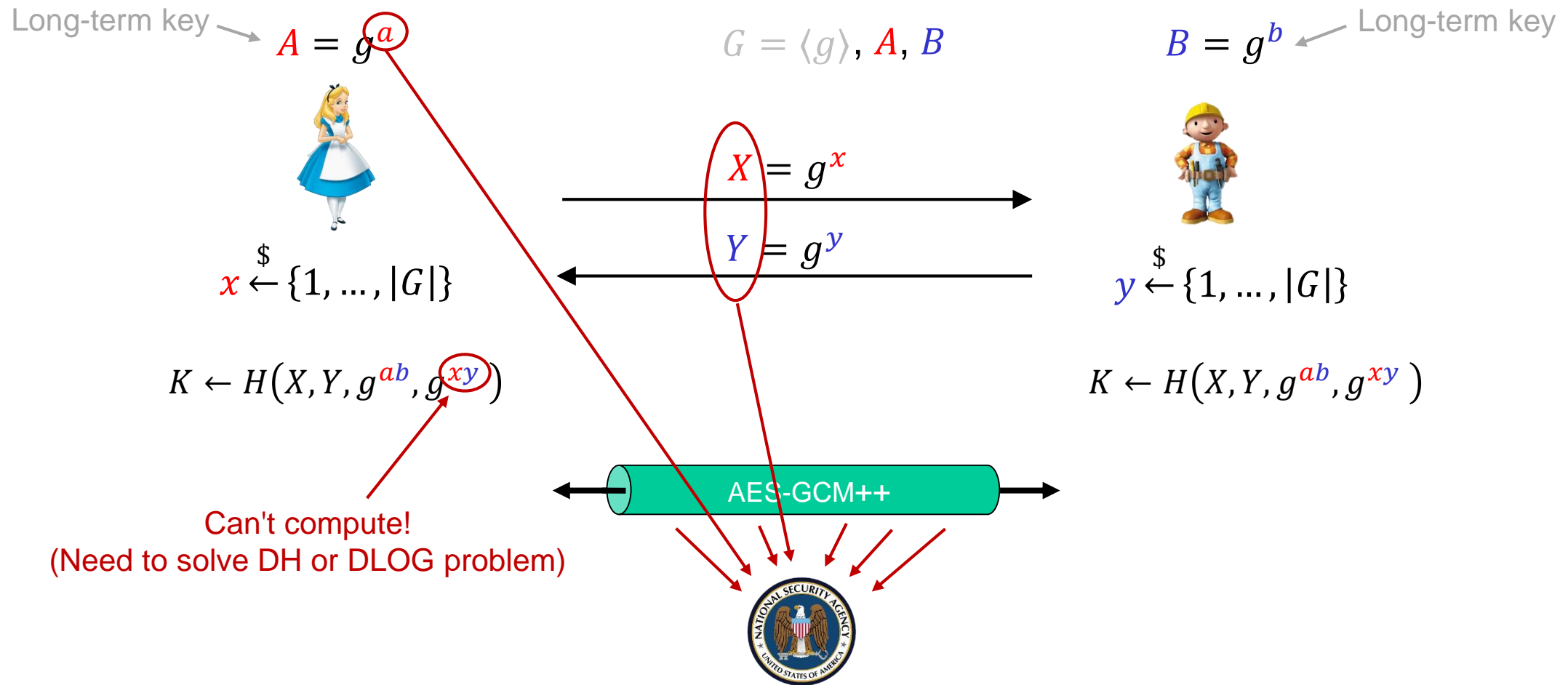
static-ephemeral, ephemeral-static, ephemeral-ephemeral

static-static, static-ephemeral, ephemeral-static, ephemeral-ephemeral

Used inside



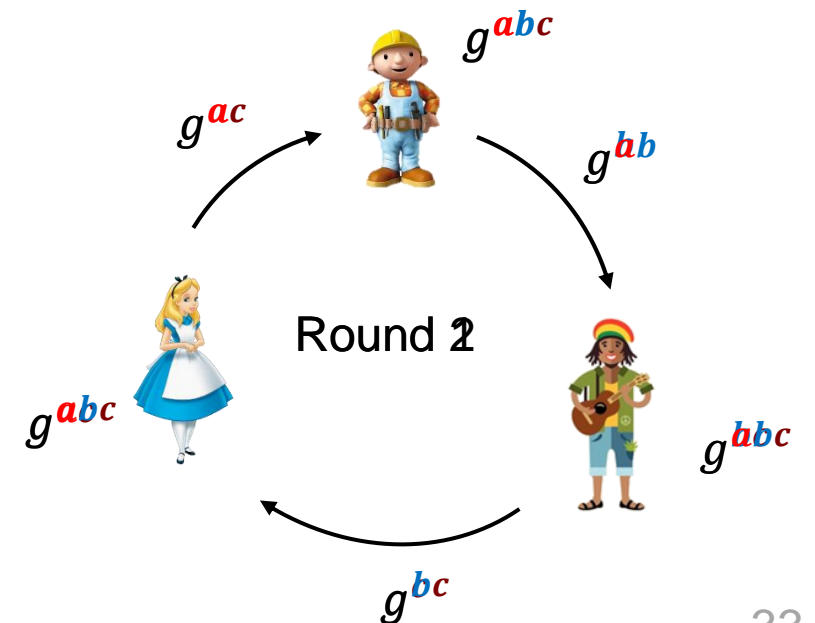
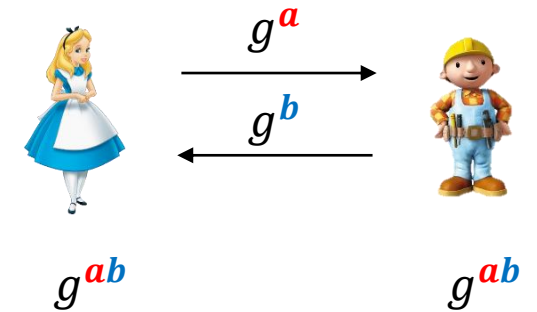
Forward secrecy



Forward secrecy: attacker can't obtain old traffic keys even if it obtains the parties' long-term keys

N-party Diffie-Hellman

- N-party Diffie-Hellman possible in $N - 1$ rounds
- 1-round N-party Diffie-Hellman:
 - $N = 2$: normal Diffie-Hellman
 - $N = 3$: Diffie-Hellman with *bilinear pairings* (Joux '00)
 - $N \geq 4$: open problem
 - Possible with *multilinear maps* (very advanced)
 - ...but we don't know any secure multilinear maps ☹️



Summary

- Special algorithms needed to deal with the large numbers in asymmetric cryptography
 - Square-and-multiply for group exponentiation
 - Straightforward implementation not secure against side-channel attacks
 - Prime finding
 - Common in practice: pick random number; check if prime
 - Primality tests: Fermat's, Miller-Rabin (small one-sided error)
- Plain Diffie-Hellman is *not* secure against active adversaries
 - Man-in-the-middle attack
 - Problem: lack of authentication
 - Common solution: digital signatures (used in TLS, IPsec, SSH)
 - Modern solution: long-term Diffie-Hellman keys mixed with ephemeral Diffie-Hellman keys
 - Noise protocol, Signal, What'sApp, Facebook Messenger Secret Conversations