Lecture 9 – Diffie-Hellman key exchange II, elliptic curves, computational aspects

TEK4500

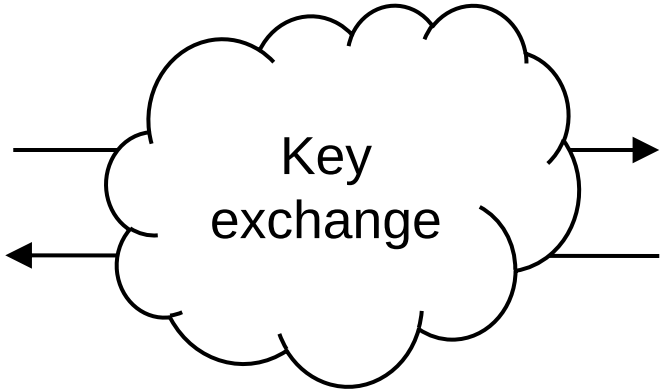
15.10.2025

Håkon Jacobsen

hakon.jacobsen@its.uio.no

Diffie-Hellman

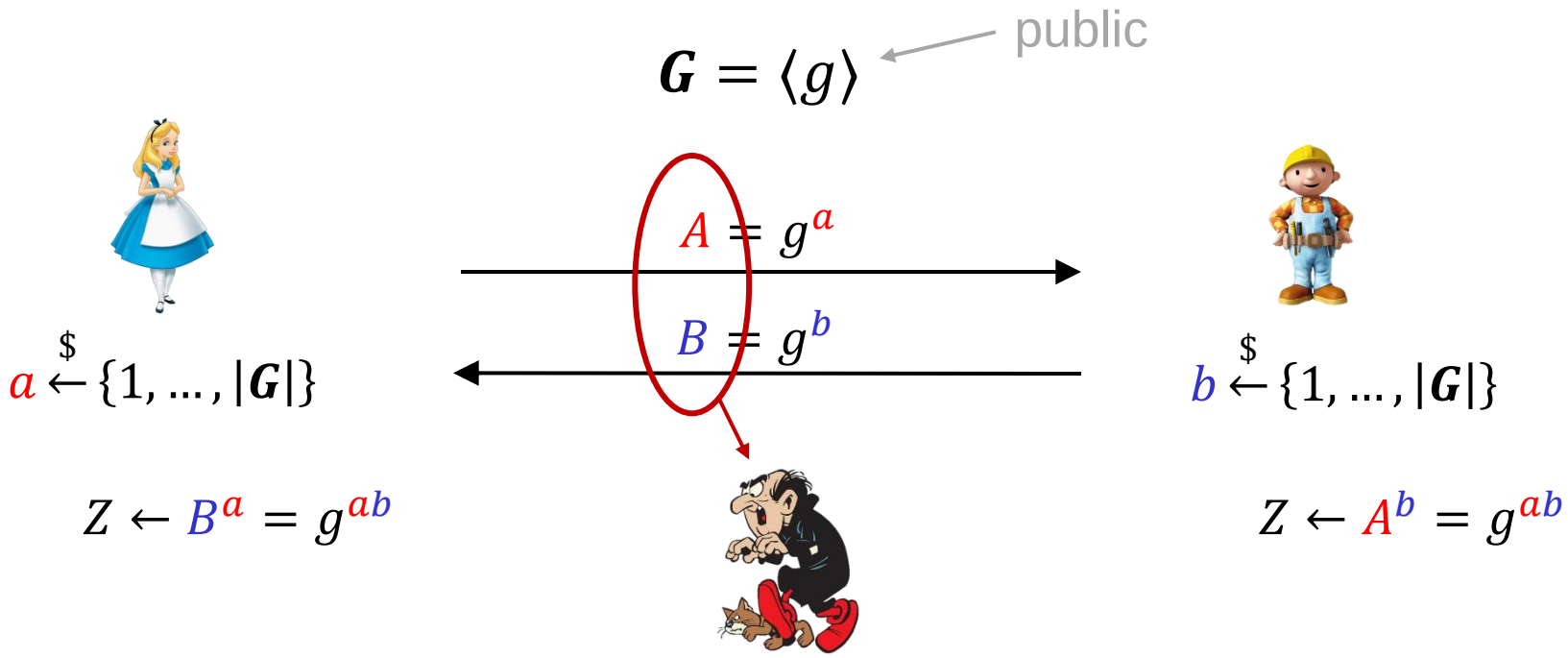








Recap: Diffie-Hellman



Security (given G, g, A, B):

- Must be hard to compute $Z \leftarrow g^{ab}$ (DH assumption)
- Must be hard to find a (or b) (DLOG assumption)

- $G = (Z_n, +)$ not secure
- $G = (Z_p^*, \cdot)$ secure for $|Z_p^*|$ **large enough** (conjectured)

Recap: generic algorithms for solving DLOG

- Baby-step, giant-step: time $\mathcal{O}\left(\sqrt{|\boldsymbol{G}|}\right)$ memory $\mathcal{O}\left(\sqrt{|\boldsymbol{G}|}\right)$
- Pollard's rho: time $\mathcal{O}\left(\sqrt{|\boldsymbol{G}|}\right)$ memory $\mathcal{O}(1)$

- **Consequence:** for DLOG to be hard $\sqrt{|\boldsymbol{G}|}$ must be large enough!
  - $|\boldsymbol{G}| \approx 2^{128}$ only gives $\sqrt{2^{128}} = 2^{64}$ security
  - $|\boldsymbol{G}| \approx 2^{256}$ only gives $\sqrt{2^{256}} = 2^{128}$ security
  - $|\boldsymbol{G}| \approx 2^{512}$ only gives $\sqrt{2^{512}} = 2^{256}$ security
  - etc...

Recap: non-generic algorithms for DLOG

- Unfortunately, $(\boldsymbol{Z}_p^*, \cdot)$ is *not* a generic group!

- *Much* faster specific algorithms exist for solving DLOG in $\boldsymbol{Z}_p^*$
  - Index-calculus
  - Elliptic-curve method
  - Special number-field sieve (SNFS)
  - General number-field sieve (GNFS)

  } *exceptionally* complicated algorithms, requiring very advanced mathematics!

- Current DLOG-solving record: $\left|\boldsymbol{Z}_p^*\right| \approx 2^{795}$ using GNFS (Heninger et al. '19)
  - Previous records: https://en.wikipedia.org/wiki/Discrete_logarithm_records

- $\left|\boldsymbol{Z}_p^*\right| \geq 2^{2048}$ typically required as a minimum today

Diffie-Hellman in $\left(Z_p^*,\cdot\right)$ – what is large enough?

$p =$ 171254583176141379301960419792575778264088323240375085733932929816426671397476217788024387752387285929683446135893799323484756135034769321631669738132186983438164632891441853629126025225404949830905314972329658295365245072698488256583114202993359222957097432675083225259667739503949192575768420387716327420441424710535098501236058838158571626669177751934961573726561955583057270098912760065140004093658772181713883199238963093777917625906143118496429613802248519404604217104493689272529748703958739363879096722748832953774810081504758785902705917983505634881680809238046118223875201980540029906239114543891047740921832147483648
$\approx 2^{2048}$



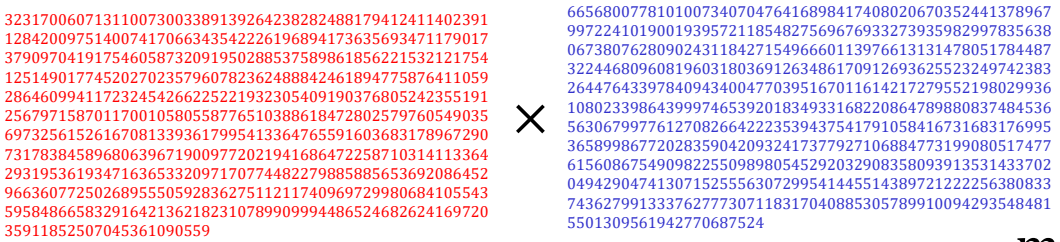
$A = 2^{\text{323170060713110073003389139264238282488179412411402391 128420097514007417066343542226196894173635693471179017 379097041917546058732091950288537589861856221532121754 125149017745202702357960782362488842461894775876411059 286460994117232454266225221932305409190376805242355191 256797158701170010580558776510388618472802579760549035 697325615261670813393617995413364765591603683178967290 731783845896806396719009772021941686472258710314113364 293195361934716365332097170774482279885885653692086452 966360772502689555059283627511211740969729980684105543 595848665832916421362182310789909994486524682624169720 359118525070453610905591 mod p$



$\xleftarrow{\$} \{1 \dots p\}$

$B = 2^{\text{665680077810100734070476416898417408020670352441378967 997224101900193957211854827569676933273935982997835638 067380762809024311842715496660113976613131478051784487 322446809608196031803691263486170912693625523249742383 264476433978409434004770395167011614217279552198029936 108023398643999746539201834933168220864789880837484536 563067997761270826642223539437541791058416731683176995 365899867720283590420932417377927106884773199080517477 615608675490982255098980545292032908358093913531433702 049429047413071525556307299541445514389721222256380833 743627991333762777307118317040885305789910094293548481 550130956194277068752 4 \mod p$

$Z \leftarrow 2^{\text{323170060713110073003389139264238282488179412411402391 128420097514007417066343542226196894173635693471179017 379097041917546058732091950288537589861856221532121754 125149017745202702357960782362488842461894775876411059 286460994117232454266225221932305409190376805242355191 256797158701170010580558776510388618472802579760549035 697325615261670813393617995413364765591603683178967290 731783845896806396719009772021941686472258710314113364 293195361934716365332097170774482279885885653692086452 966360772502689555059283627511211740969729980684105543 595848665832916421362182310789909994486524682624169720 359118525070453610905591} \times$



$\mod p$



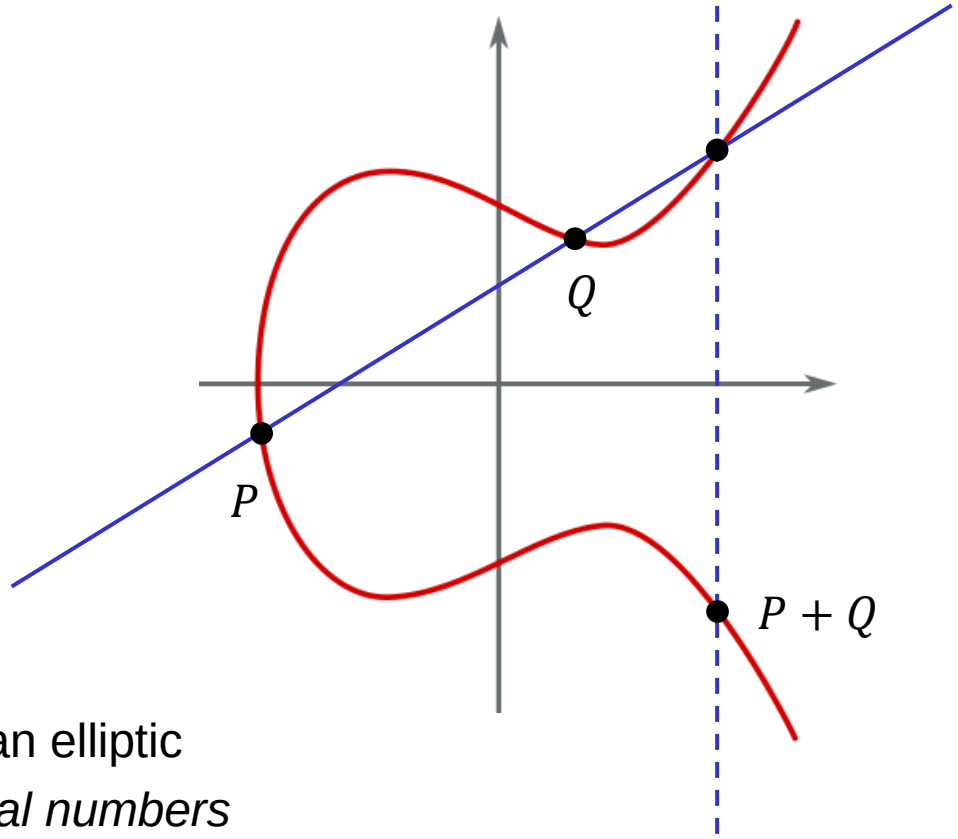
Z_p^*

Groups where GNFS doesn't work

Better alternatives to Z_p^*?

Elliptic curves

Theorem: the points on an elliptic curve, together with $\mathcal{O}$, is an abelian group under "geometric point addition"



$$y^2 = x^3 + ax + b$$

$$a, b, x, y \in \boldsymbol{R}$$

$$P + Q = (x_3, y_3)$$

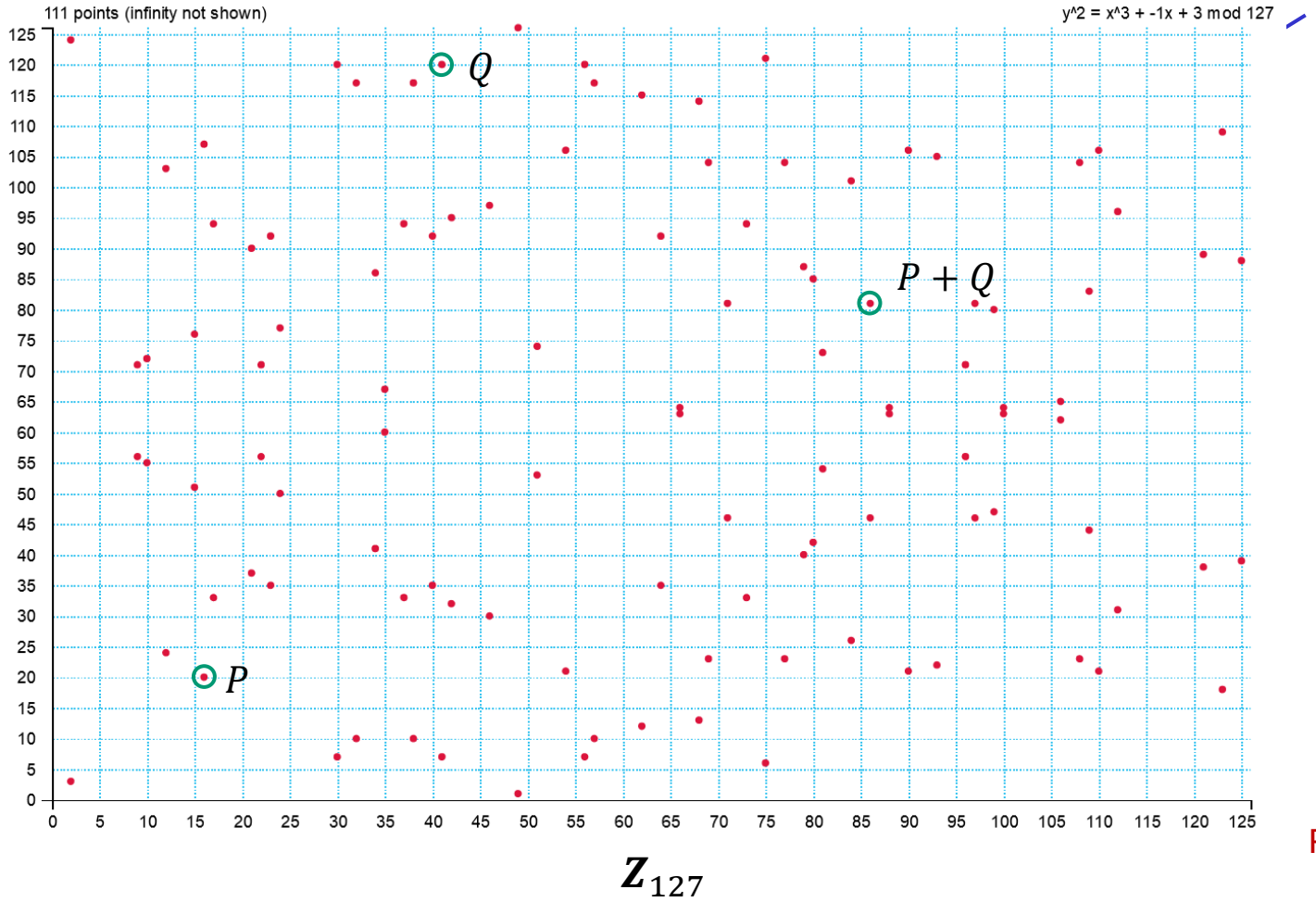
$$x_3 = \frac{(x_1x_2 - 2a)x_1x_2 - 4b(x_1 + x_2) + a^2}{(x_1x_2 + a)(x_1 + x_2) + 2y_1y_2 + 2b}$$

$$y_3 = \frac{x_1x_2(x_1 + x_2) - x_3\big((x_1 + x_2)^2 - x_1x_2 + a\big) - y_1y_2 - b}{y_1 + y_2}$$

Notation: $(\boldsymbol{E}(\boldsymbol{R}), +) =$ an elliptic curve group over the *real numbers*

Elliptic curves over Z_p

Theorem: the points in Z_p satisfying $y^2 = x^3 + ax + b$, together with $\mathcal{O}$, is an abelian group under "geometric point addition"



Z_{127}

Notation: $(E(F_p), +)$

$$y^2 = x^3 + ax + b$$

$$a, b, x, y \in Z_p$$

$$P + Q = (x_3, y_3)$$

Still valid!

$$x_3 = \frac{(x_1x_2 - 2a)x_1x_2 - 4b(x_1 + x_2) + a^2}{(x_1x_2 + a)(x_1 + x_2) + 2y_1y_2 + 2b}$$

$$y_3 = \frac{x_1x_2(x_1 + x_2) - x_3\left((x_1 + x_2)^2 - x_1x_2 + a\right) - y_1y_2 - b}{y_1 + y_2}$$

$(Z_p, +)$ $(Z_p^*, \cdot)$

$(F_p, +, \cdot)$

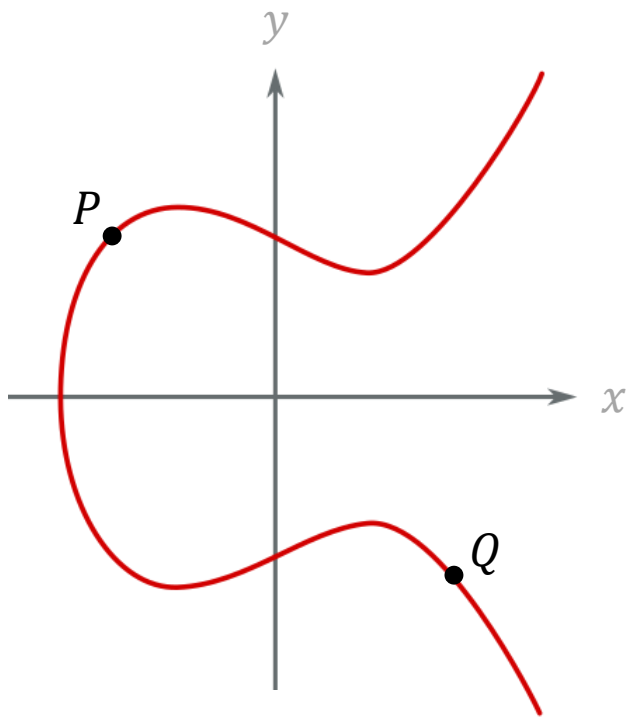
Finite field

Elliptic curves summary

- $P = (x, y)$ point on elliptic curve
- x and y are elements in $\boldsymbol{Z}_p$
- Points P and Q can be added to get another point $P + Q$
- Special case: add P to itself n times

$$nP = P + P + \cdots + P$$

- DLOG problem: given P and nP, find n (for **secret** n)



$E(Z_p)$ – properties

- $(Z_p^*, \cdot)$ is *not* a generic group
  - Specialized attacks (GNFS) exploit algebraic structure ⟹ parameters must be bigger to compensate
  - $|Z_p^*| \geq 2^{2048}$ required for security today
  - Bigger parameters ⟹ slower systems

- Currently no attacks manage to exploit the algebraic structure of $(E(Z_p), +)$
  - Best-know attacks are *generic attacks* which require time $\Omega\left(\sqrt{|G|}\right)$
  - Consequently: elliptic curve crypto can use *much* smaller parameters
  - $|E(Z_p)| = 2^{256}, 2^{384}, 2^{512}$ common in practice ⟹ much faster than Z_p^*-based crypto

Diffie-Hellman in $\left(Z_p^*, \cdot\right)$

$p =$ 171254583176141379301960419792575778264088323240375085733932929816426671397476217788024387752387285929683446135893799323484756135034769321631669738132186983438164632891441853629126025225404949830905314972329658295365245072698488256583114202993359222957097432675083225259667739503949192575768420387716327420441424710535098501236058838158571626669177751934961573726561955583057270098912760065140004093658772181713883199238963093777917625906143118496429613802248519404604217104493689272529748703958739363879096722748832953774810081504758785902705917983505634881680809238046118223875201980540029906239114543891047740921832147483648 $\approx 2^{2048}$



$A = 2$ mod p

323170060713110073003389139264238282488179412411402391
128420097514007417066343542226196894173635693471179017
379097041917546058732091950288537589861856221532121754
125149017745202702357960782362488842461894775876411059
286460994117232454266225221932305409190376805242355191
256797158701170010580558776510388618472802579760549035
697325615261670813393617995413364765591603683178967290
731783845896806396719009772021941686472258710314113364
293195361934716365332097170774482279885885653692086452
966360772502689555059283627511211740969729980684105543
595848665832916421362182310789909994486524682624169720
359118525070453610905591



$\xleftarrow{\$} \{1 \dots p\}$

$B = 2$ mod p

665680077810100734070476416898417408020670352441378967
997224101900193957211854827569676933273935982997835638
067380762809024311842715496660113976613131478051784487
322446809608196031803691263486170912693625523249742383
264476433978409434004770395167011614217279552198029936
108023398643999746539201834933168220864789880837484536
563067997761270826642223539437541791058416731683176995
365899867720283590420932417377927106884773199080517477
615608675490982255098980545292032908358093913531433702
049429047413071525556307299541445514389721222256380833
743627991333762777307118317040885305789910094293548481
550130956194277068752 4

$\xrightarrow{\$} \{1 \dots p\}$

665680077810100734070476416898417408020670352441378967
997224101900193957211854827569676933273935982997835638
067380762809024311842715496660113976613131478051784487
322446809608196031803691263486170912693625523249742383
264476433978409434004770395167011614217279552198029936
108023398643999746539201834933168220864789880837484536
563067997761270826642223539437541791058416731683176995
365899867720283590420932417377927106884773199080517477
615608675490982255098980545292032908358093913531433702
049429047413071525556307299541445514389721222256380833
743627991333762777307118317040885305789910094293548481
550130956194277068752 4

$Z \leftarrow 2$ mod p

$\times$

323170060713110073003389139264238282488179412411402391
128420097514007417066343542226196894173635693471179017
379097041917546058732091950288537589861856221532121754
125149017745202702357960782362488842461894775876411059
286460994117232454266225221932305409190376805242355191
256797158701170010580558776510388618472802579760549035
697325615261670813393617995413364765591603683178967290
731783845896806396719009772021941686472258710314113364
293195361934716365332097170774482279885885653692086452
966360772502689555059283627511211740969729980684105543
595848665832916421362182310789909944865246826241697 20
359118525070453610905591

665680077810100734070476416898417408020670352441378967
997224101900193957211854827569676933273935982997835638
067380762809024311842715496660113976613131478051784487
322446809608196031803691263486170912693625523249742383
264476433978409434004770395167011614217279552198029936
108023398643999746539201834933168220864789880837484536
563067997761270826642223539437541791058416731683176995
365899867720283590420932417377927106884773199080517477
615608675490982255098980545292032908358093913531433702
049429047413071525556307299541445514389721222256380833
743627991333762777307118317040885305789910094293548481
550130956194277068752 4

$\text{mod } p$

Diffie-Hellman in $(E(Z_p), +)$

$p = 115792089210356248762697446949407573530086143415290314195533631308867097853951 \approx 2^{256}$



$A = 89080287395774076886404465276262680764297347818898641719959470631357066012729 \cdot Q$



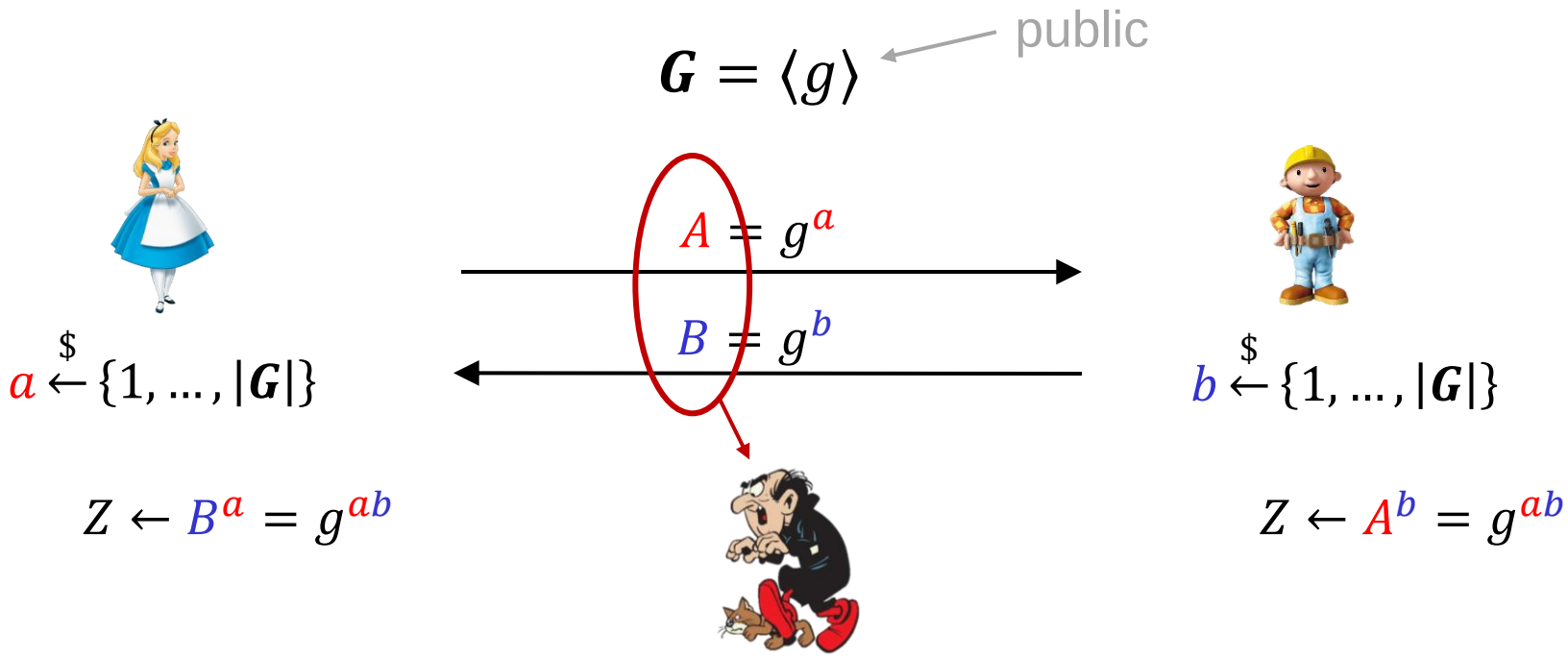
$89080287395774076886404465276262680764297347818898641719959470631357066012729 \xleftarrow{\$} \{1 \dots p\}$

$47777709944406917119610162988578419285974852857762532427086507801748071058272 \xleftarrow{\$} \{1 \dots p\}$

$B = 47777709944406917119610162988578419285974852857762532427086507801748071058272 \cdot Q$

$Z \leftarrow 89080287395774076886404465276262680764297347818898641719959470631357066012729 \times 47777709944406917119610162988578419285974852857762532427086507801748071058272 \cdot Q$

Diffie-Hellman – security



Security (given G, g, A, B):

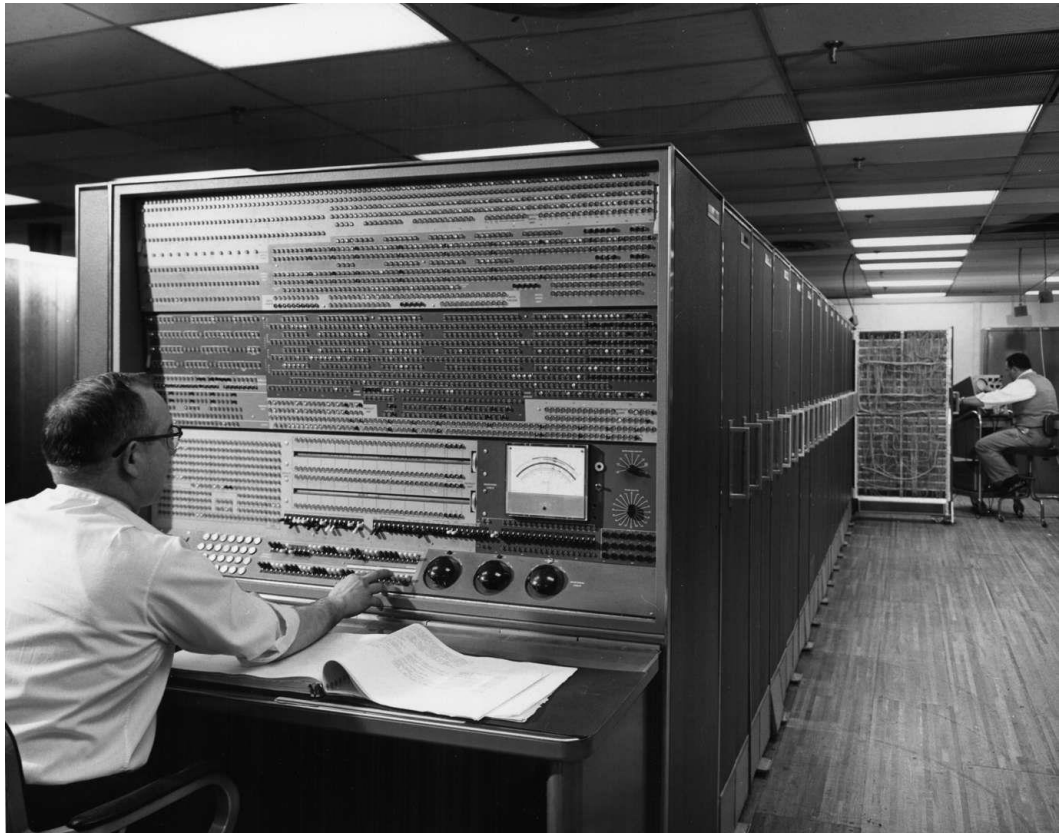
- Must be hard to compute $Z \leftarrow g^{ab}$ (DH assumption)
- Must be hard to find a (or b) (DLOG assumption)

- $G = (Z_p, +)$ not secure
- $G = (Z_p^*, \cdot)$ secure for $|Z_p^*|$ **large enough** (conjectured)
- $G = (E(F_p), +)$ secure for $|E(F_p)|$ **large enough** (conjectured)

Cryptographic groups in practice

- $\left(\boldsymbol{Z}_p^*, \cdot\right)$
  - **TLS 1.3:** five specific groups allowed
    - size $\approx 2^{2048}, 2^{3072}, 2^{4096}, 2^{6144}, 2^{8192}$ (RFC 7919)
  - **IKEv2** (IPsec key exchange protocol): MODP groups
    - size $\approx 2^{768}, 2^{1024}, 2^{1536}, 2^{2048}, 2^{3072}, 2^{4096}, 2^{6144}, 2^{8192}$ (RFC 7296 and RFC 3526)

- $\left(\boldsymbol{E}\left(\boldsymbol{Z}_p^*\right), +\right)$
  - NIST curves: P-224, P-256, P-384, P-521
  - Curve25519 ($E: y^2 = x^3 + 486662x^2 + x$ and $p = 2^{255} - 19$) (Daniel J. Bernstein)
  - Curve448 ($E: y^2 + x^2 = 1 - 39081x^2y^2$ and $p = 2^{448} - 2^{224} - 1$) (Mike Hamburg)



Computational aspects

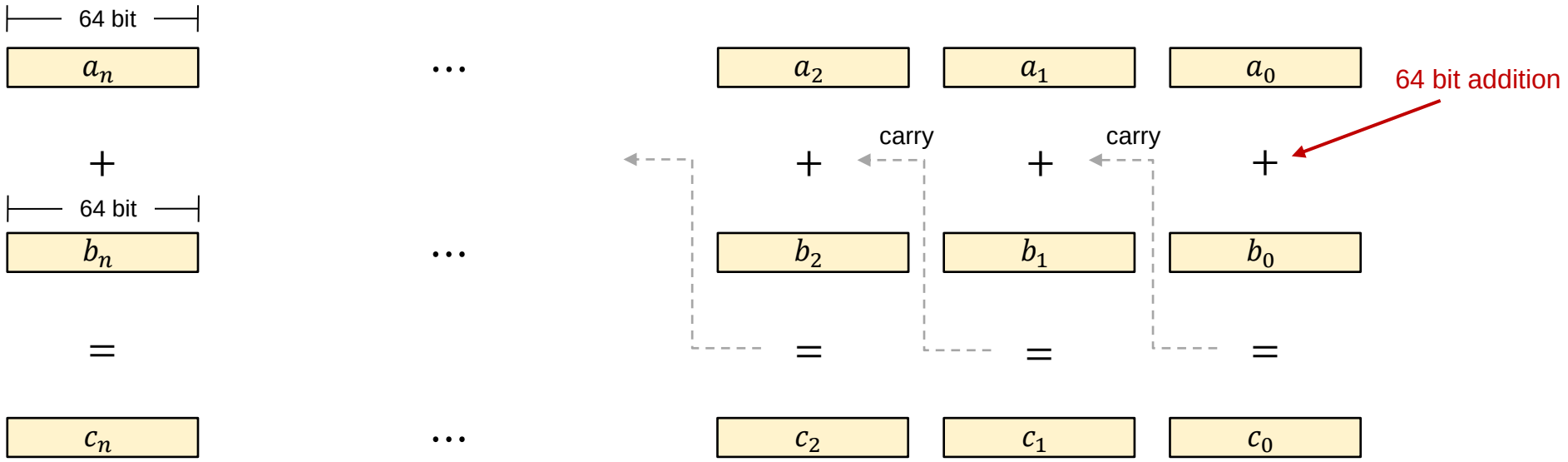
Big-number arithmetic

$a =$ 32317006071311007300338913926423828248817941241140239112842009751400741706634354222619689417363569347117901737909704191754605873209195028853758986185622153212175412514901774520270235796078236248884246189477587641105928646099411723245426622522193230540919037680524235519125679715870117001058055877651038861847280257976054903569732561526167081339361799541336476559160368317896729073178384589680639671900977202194168647225871031411336429319536193471636533209717077448227988588565369208645296636077250268955505928362751121174096972998068410554359584866583291642136218231078990999448652468262416972035911852507045361090559

$b =$ 66568007781010073407047641689841740802067035244137896799722410190019395721185482756967693327393598299783563806738076280902431184271549666011397661313147805178448732244680960819603180369126348617091269362552324974238326447643397840943400477039516701161421727955219802993610802339864399974653920183493316822086478988083748453656306799776127082664222353943754179105841673168317699536589986772028359042093241737792710688477319908051747761560867549098225509898054529203290835809391353143370204942904741307152555630729954144551438972122225638083374362799133376277730711831704088530578991009429354848155013095619427706875 24

$a + b = ?$

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



Big-number arithmetic

$a =$ 32317006071311007300338913926423828248817941241140239112842009751400741706634354222619689417363569347117901737909704191754605873209195028853758986185622153212175412514901774520270235796078236248884246189477587641105928646099411723245426622522193230540919037680524235519125679715870117001058055877651038861847280257976054903569732561526167081339361799541336476559160368317896729073178384589680639671900977202194168647225871031411336429319536193471636533209717077448227988588565369208645296636077250268955505928362751121174096972998068410554359584866583291642136218231078990999448652468262416972035911852507045361090559

$b =$ 66568007781010073407047641689841740802067035244137896799722410190019395721185482756967693327393598299783563806738076280902431184271549666011397661313147805178448732244680960819603180369126348617091269362552324974238326447643397840943400477039516701161421727955219802993610802339864399974653920183493316822086478988083748453656306799776127082664222353943754179105841673168317699536589986772028359042093241737792710688477319908051747761560867549098225509898054529203290835809391353143370204942904741307152555630729954144551438972122225638083374362799133376277730711831704088530578991009429354848155013095619427706875 24

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

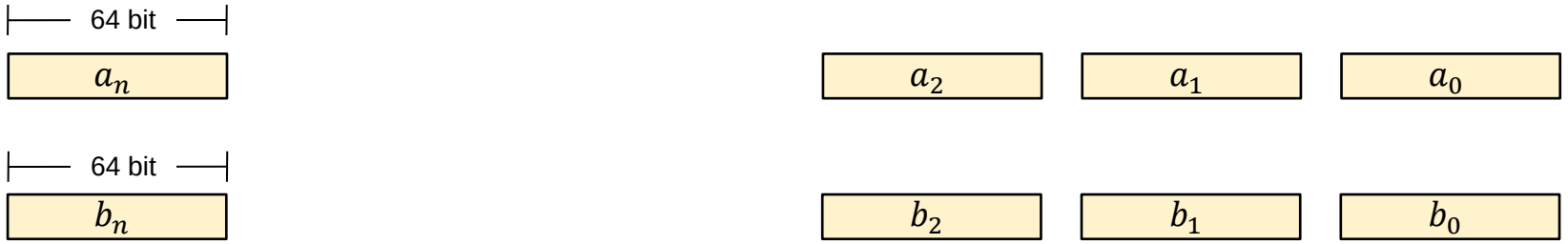
$a \cdot b = ?$

				...	carry	carry	
a_n	a_{n-1}	$\cdots$	a_3	a_2	a_1	a_0	
b_n	b_{n-1}	$\cdots$	b_3	b_2	b_1	b_0	
a_nb_0	$a_{n-1}b_0$	$\cdots$	a_3b_0	a_2b_0	a_1b_0	a_0b_0	
a_nb_1	$a_{n-1}b_1$		$\cdots$	a_2b_1	a_1b_1	a_0b_1	
a_nb_2	$a_{n-1}b_2$		$\cdots$	a_2b_2	a_1b_2	a_0b_2	
$\vdots$							
$+$ a_nb_n	$a_{n-1}b_n$	$\cdots$	a_2b_n	a_1b_n	a_0b_n		
$=$ c_{2n}	c_{2n-1}	$\cdots$	c_3	c_2	c_1	c_0	

32 bit multiplication

64 bit addition

Big-number arithmetic



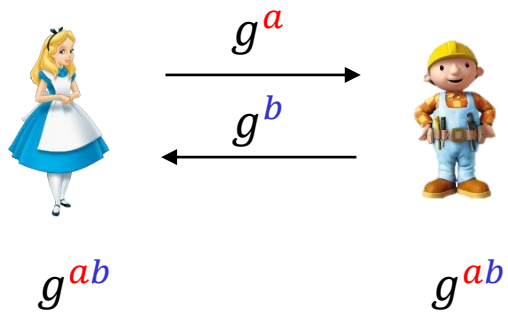
Algorithm	Input	Output	Time
ADD	a, b	$a + b$	$\mathcal{O}(n)$
MULT	a, b	ab	$\mathcal{O}(n^2)$
MOD	a, p	$a \bmod p$	$\mathcal{O}(n^2)$
MOD-INV	a, p	$a^{-1} \bmod p$	$\mathcal{O}(n^2)$

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

* $n > 2^{7137398073256634897664758526207831206 41}$

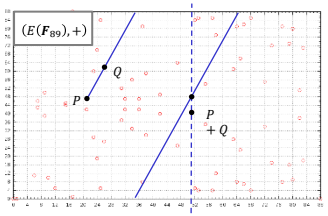
Diffie-Hellman – computations

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



Group exponentiation g^a

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



Computing in groups – exponentiation

Goal: compute $g^n \in \boldsymbol{G}$

Naïve approach: $g^n = \overbrace{g \star g \star g \star \cdots \star g}^{n}$

- Running time: $\mathcal{O}(n)$ group operations

- $\left(\boldsymbol{Z}_p^*, \cdot\right)$: $n \approx 2^{2048}$
- $\left(\boldsymbol{E}\left(\boldsymbol{F}_p\right), +\right)$: $n \approx 2^{256}$

$$\overbrace{2 \cdot 2 \cdot \quad \cdots \quad \cdot 2}^{\text{huge}} \pmod p$$

$$\underbrace{Q + Q + \quad \cdots \quad + Q}_{\text{huge}}$$

- Naïve approach is impossible in practice!

Computing in groups – exponentiation; square-and-multiply

Goal: compute $g^n \in \boldsymbol{G}$

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

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

g^{43}

$43 = 101011_2$

$$y \leftarrow e$$
$$y \leftarrow y^2 \star g^1 \qquad = g^1$$
$$y \leftarrow y^2 \star g^0 \qquad = g^2$$
$$y \leftarrow y^2 \star g^1 \qquad = g^{4+1} = g^5$$
$$y \leftarrow y^2 \star g^0 \qquad = g^{10}$$
$$y \leftarrow y^2 \star g^1 \qquad = g^{20+1} = g^{21}$$
$$y \leftarrow y^2 \star g^1 \qquad = g^{42+1} = \underline{g^{43}}$$

Computing in groups – exponentiation; square-and-multiply

Goal: compute $g^n \in \boldsymbol{G}$

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

$y \leftarrow e$

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

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

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

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

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

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

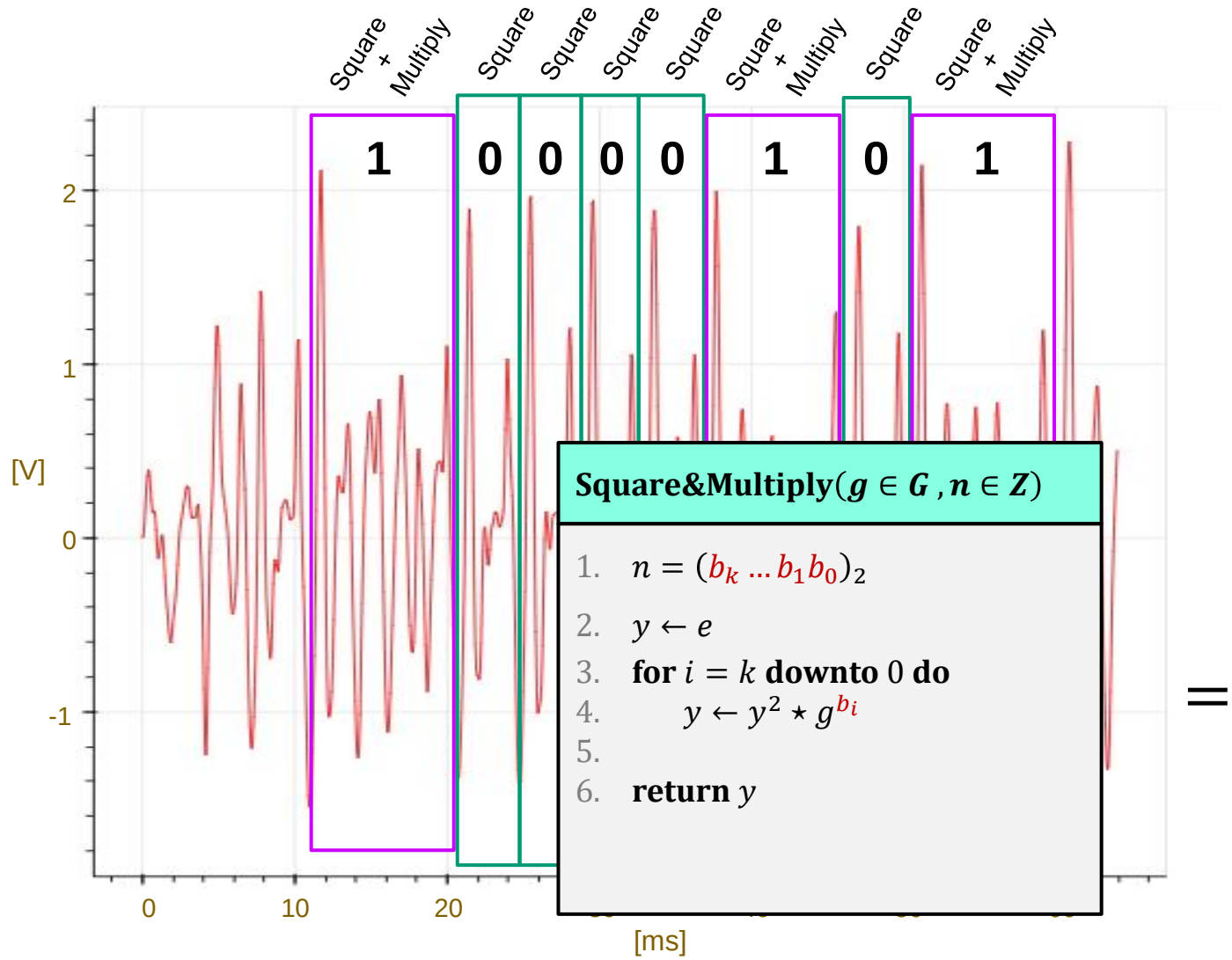
2^{2048} vs. 2048 multiplications for $(\boldsymbol{Z}_p^*, \cdot)$

Running time: $\mathcal{O}(\log_2 n)$ group operations

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

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

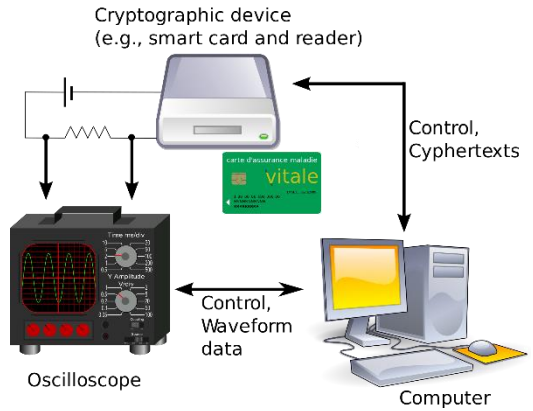
Side-channel attacks – power analysis



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

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

=



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

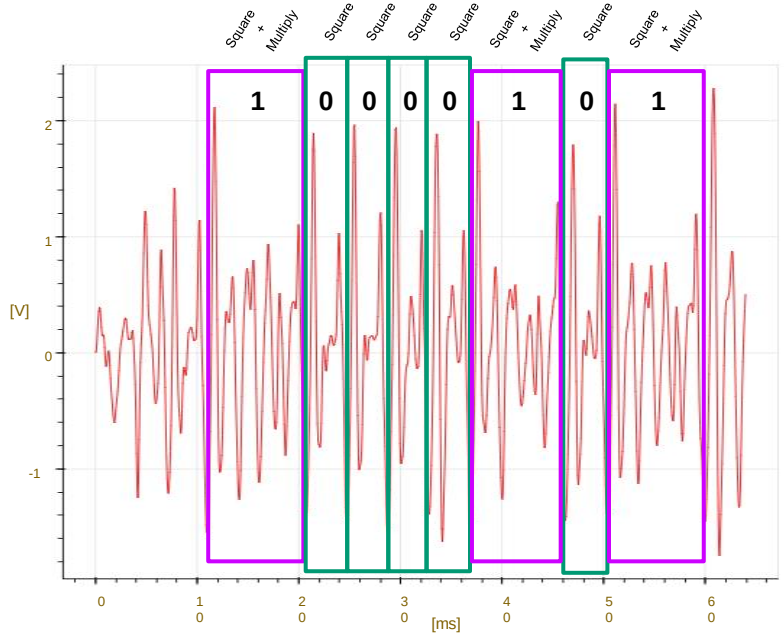
1. $n = (b_k \dots b_1 b_0)_2$
2. $y \leftarrow e$
3. **for** $i = k$ **downto** 0 **do**
4. $\quad y \leftarrow y^2$
5. $\quad$ **if** $b_i = 1$ **then**
6. $\quad\quad y \leftarrow y \star g$
7. **return** y

Mitigations

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

Square-and-*always*-Multiply$(g \in G, n \in Z)$

1. $n = (b_k \dots b_1 b_0)_2$
2. $y \leftarrow e$
3. **for** $i = k$ **downto** 0 **do**
4. $\quad y \leftarrow y^2$
5. $\quad z \leftarrow y \star g$ // always perform multiplication
6. $\quad$ **if** $b_i = 1$ **then**
7. $\quad\quad y \leftarrow z$
8. **return** y



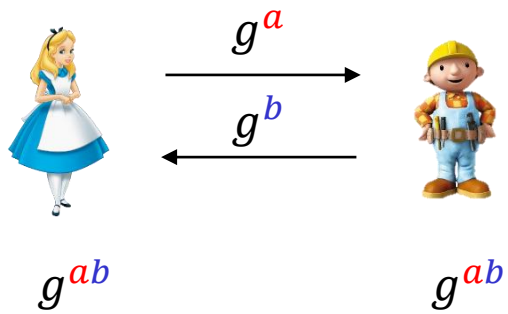
- Approximately half of the multiplications thrown away
- Many optimizations exist

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

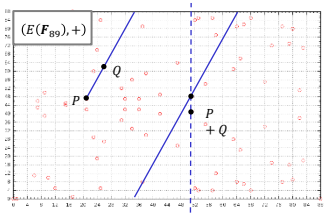
1. $n = (b_k \dots b_1 b_0)_2$
2. $y \leftarrow e$
3. **for** $i = k$ **downto** 0 **do**
4. $\quad y \leftarrow y^2$
5. $\quad$ **if** $b_i = 1$ **then**
6. $\quad\quad y \leftarrow y \star g$
7. **return** y

Diffie-Hellman – computations

- $\left(\boldsymbol{Z}_p^*, \cdot\right)$
  - Find prime p (one-time)
  - Find generator $\langle g \rangle = \left(\boldsymbol{Z}_p^*, \cdot\right)$ (one-time)
  - Compute $g^a = gg \cdots g \mod p$ (every DH exchange)
    - Multiply $x \cdot y \mod p$



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



Finding large prime numbers

- We need *large* prime numbers for both $(\boldsymbol{Z}_p^*, \cdot)$ and $(\boldsymbol{E}(\boldsymbol{F}_p), +)$
- No efficient "prime-generating" formula is known
- Simple idea: pick number at random and check if it's prime
- Efficient? $\Longrightarrow$ What is the success probability?
  - Depends on the ratio primes / non-primes
  - $\pi(n) \stackrel{\text{def}}{=}$ "#prime numbers $\leq n$"
  - **Prime Number Theorem:** $\frac{\pi(n)}{n} \approx \frac{1}{\ln n}$ $\qquad n \approx 2^{2048} \Longrightarrow \frac{\pi(n)}{n} \approx \frac{1}{\ln 2^{2048}} \approx \frac{1}{1400}$
  - 2048 bit number: ≈ 700 trials needed

GenPrime(k)

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

n	100	1000	10^6	10^9	10^{20}	10^{28}
$\pi(n)$	25	168	$\approx 10^5$	$\approx 10^8$	$\approx 10^{18}$	$\approx 10^{26}$

Finding large prime numbers

- GenPrime efficient?
  - prime-to-non-prime ratio ✓
  - $\text{IsPrime}(n)$?

- Naïve $\text{IsPrime}(n)$
  - 3 divides n? Not prime
  - 5 divides n? Not prime
  - 7 divides n? Not prime
  - ⋮
  - $\lfloor \sqrt{n} \rfloor$ divides n? Not prime
  - n is prime!

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

- 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** $\text{IsPrime}(n)$ **then**
4. **return** n

Fermat's theorem

Theorem: if $(\boldsymbol{G},\star)$ is a finite group, then for all $g \in \boldsymbol{G}$:

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

$$\left(\boldsymbol{Z}_p^*,\cdot\right) = \{1,2,\dots,p-1\} \qquad \left|\boldsymbol{Z}_p^*\right| = p-1$$

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

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

A B

p is prime $\implies$ *all* $a \neq 0 \bmod p$ satisfy Fermat's theorem

p is **not** prime $\impliedby$ *some* $a \neq 0 \bmod p$ does **not** satisfy Fermat's theorem

$\bar{A}$ $\bar{B}$

Fermat primality test

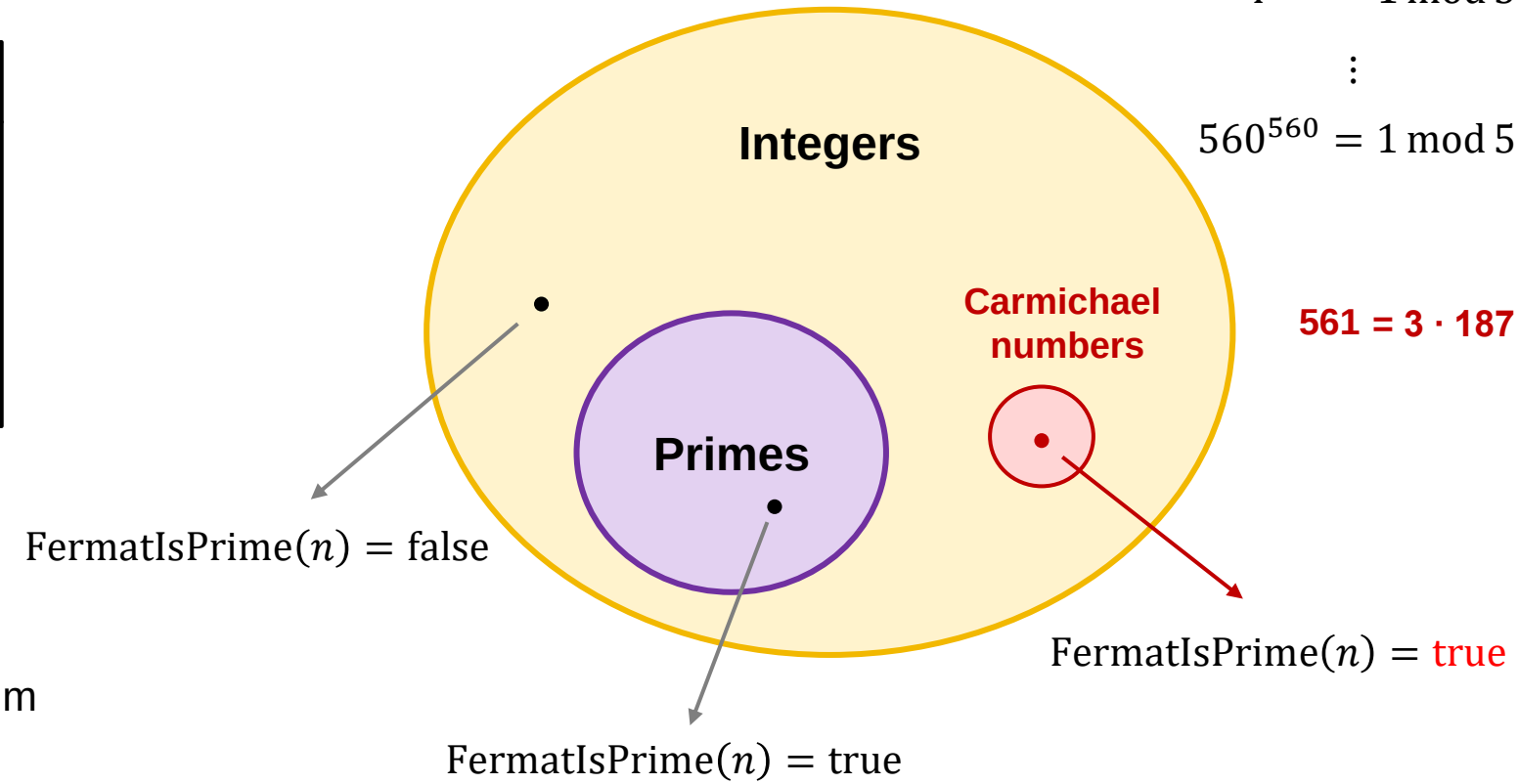
Fermat's theorem: if p is prime, then for all $a \neq 0 \pmod p$:

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

$$2^{560} = 1 \bmod 561$$
$$3^{560} = 1 \bmod 561$$
$$4^{560} = 1 \bmod 561$$
$$\vdots$$
$$560^{560} = 1 \bmod 561$$

FermatIsPrime(n)

1. **for** $a \in \{2, 3, \ldots, t\}$ **do**
2. **if** $a^{n-1} \neq 1 \bmod n$ **then**
3. **return** false
4. **return** true



Better tests exist:

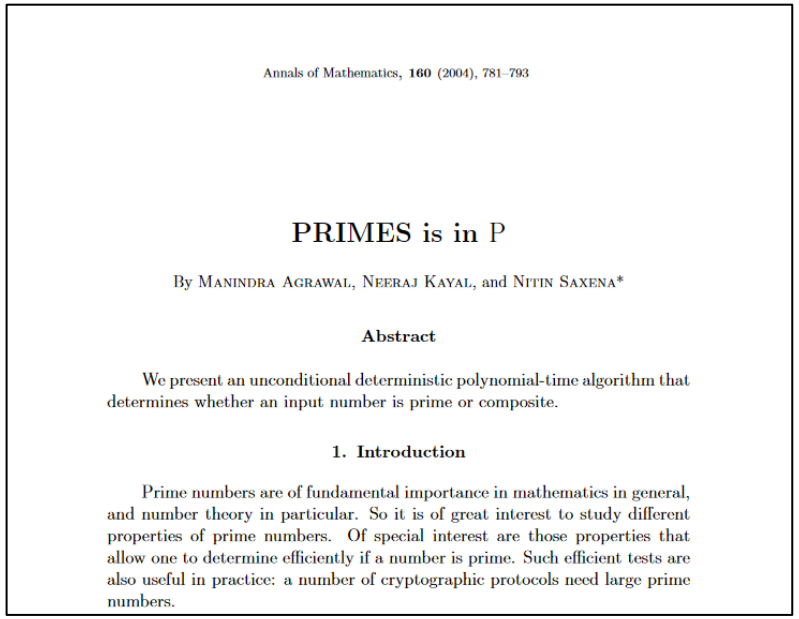
- Solovay-Strassen algorithm
- Miller-Rabin algorithm

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 $\approx \mathcal{O}(k^{12})$
  - Quickly improved to $\approx \mathcal{O}(k^6)$

MillerRabinIsPrime(n)

1. **for** $a \in \{2, 3, \ldots, t\}$ **do**
2. **if** MillerRabinTest(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

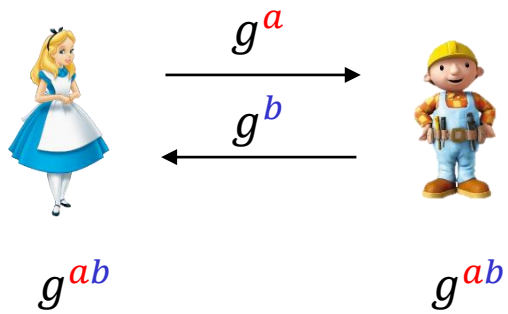
We present an unconditional deterministic polynomial-time algorithm that determines whether an input number is prime or composite.

1. Introduction

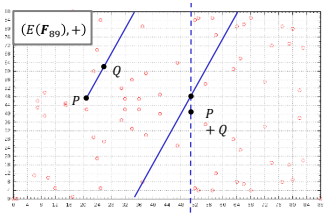
Prime numbers are of fundamental importance in mathematics in general, and number theory in particular. So it is of great interest to study different properties of prime numbers. Of special interest are those properties that allow one to determine efficiently if a number is prime. Such efficient tests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

Diffie-Hellman – computations

- $\left(\boldsymbol{Z}_p^*, \cdot\right)$
  - Find prime p (one-time)
  - Find generator $\langle g \rangle = \left(\boldsymbol{Z}_p^*, \cdot\right)$ (one-time) Easy
  - Compute $g^a = gg \cdots g \mod p$ (every DH exchange)
    - Multiply $x \cdot y \mod p$

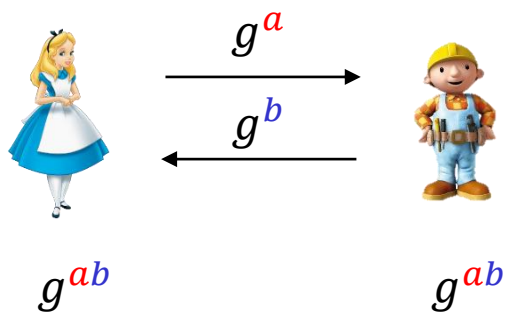


- $\left(\boldsymbol{E}\left(\boldsymbol{F}_p\right), +\right)$
  - Find prime p (one-time)
  - Generate curve $y^2 = x^3 + ax + b \mod p$ (one-time) Tricky! Standardized choices (NIST P-256, Curve25519)
  - Find curve group order $\left|E\left(\boldsymbol{F}_p\right)\right|$ (one-time) Tricky! (Schoof's algorithm) $\left|\boldsymbol{E}\left(\boldsymbol{F}_p\right)\right| \neq p$ (typically)
  - Find generator Q (one-time) Easy! $\left|\boldsymbol{E}\left(\boldsymbol{F}_p\right)\right|$ usually prime (or close to prime)
  - Compute $aQ = Q + Q + \cdots + Q$ (every DH exchange)
    - Point addition

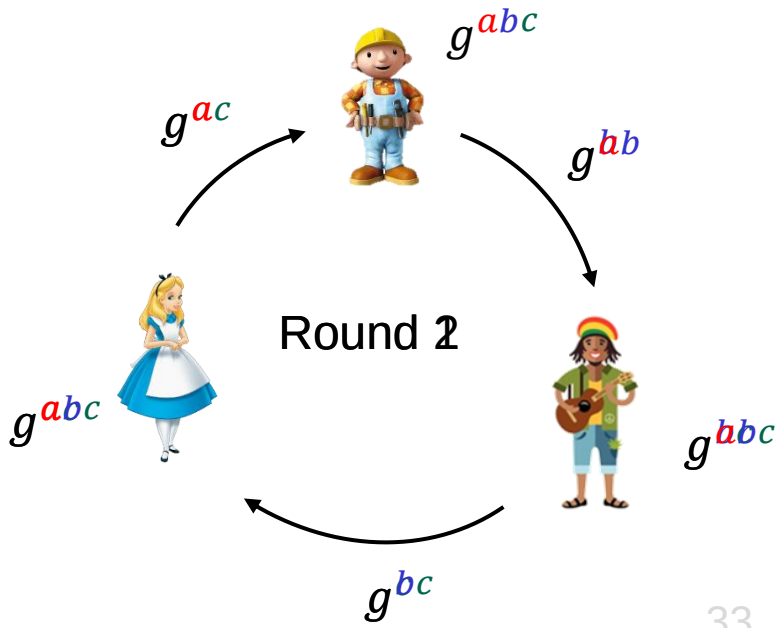


N-party Diffie-Hellman

- 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*



Next week

- More Diffie-Hellman
- Public-Key Encryption/Key Encapsulation