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

TEK4500

23.10.2024

Håkon Jacobsen

hakon.jacobsen@its.uio.no

Summary of group theory

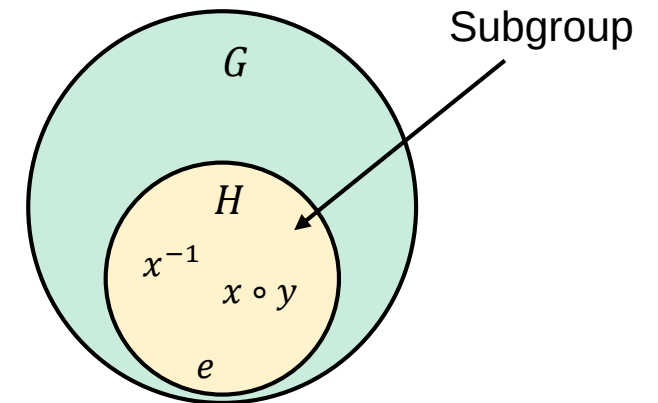
- Groups
- Subgroups
- Cyclic groups
 - Groups generated by one element $g \in G$
 - $(G, \circ) = \langle g \rangle$
 - Examples: $(\mathbf{Z}, +) = \langle 1 \rangle$, $(\mathbf{Z}_n, +) = \langle 1 \rangle$, $(\mathbf{Z}_p^*, \cdot)$
- Properties:
 - $g^{|G|} = e$
 - $g^a g^b = g^{a+b}$
 - $(g^a)^b = g^{ab} = (g^b)^a$

Definition: A **group** $(G, \circ)$ is a set G with a binary operation $\circ$ where

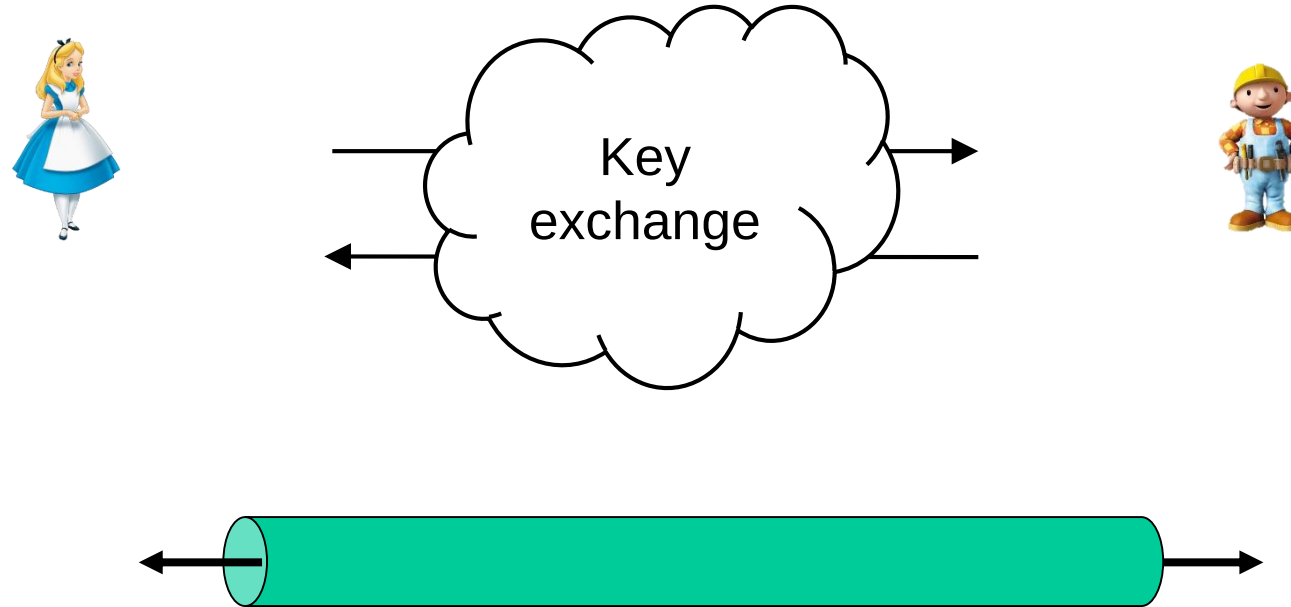
$\mathcal{G}1$: $(a \circ b) \circ c = a \circ (b \circ c)$ (associativity)

$\mathcal{G}2$: $\exists e \in G: e \circ a = a \circ e = a$ (identity)

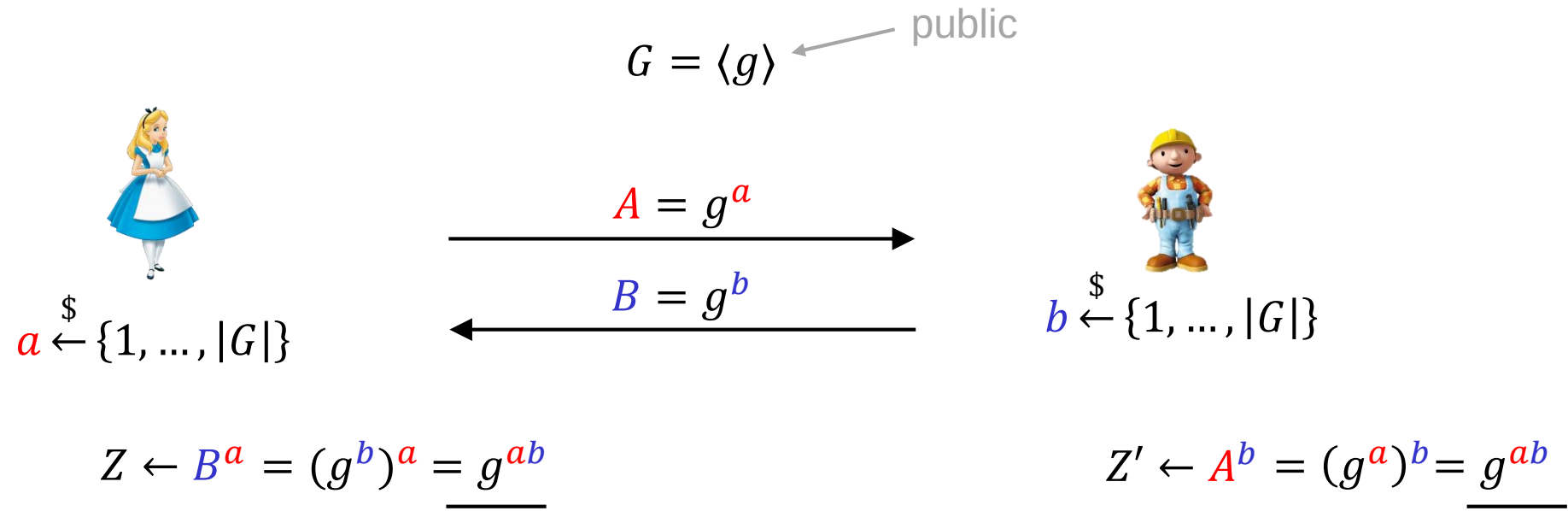
$\mathcal{G}3$: $\exists a^{-1} \in G: a \circ a^{-1} = a^{-1} \circ a = e$ (inverse)



Diffie-Hellman

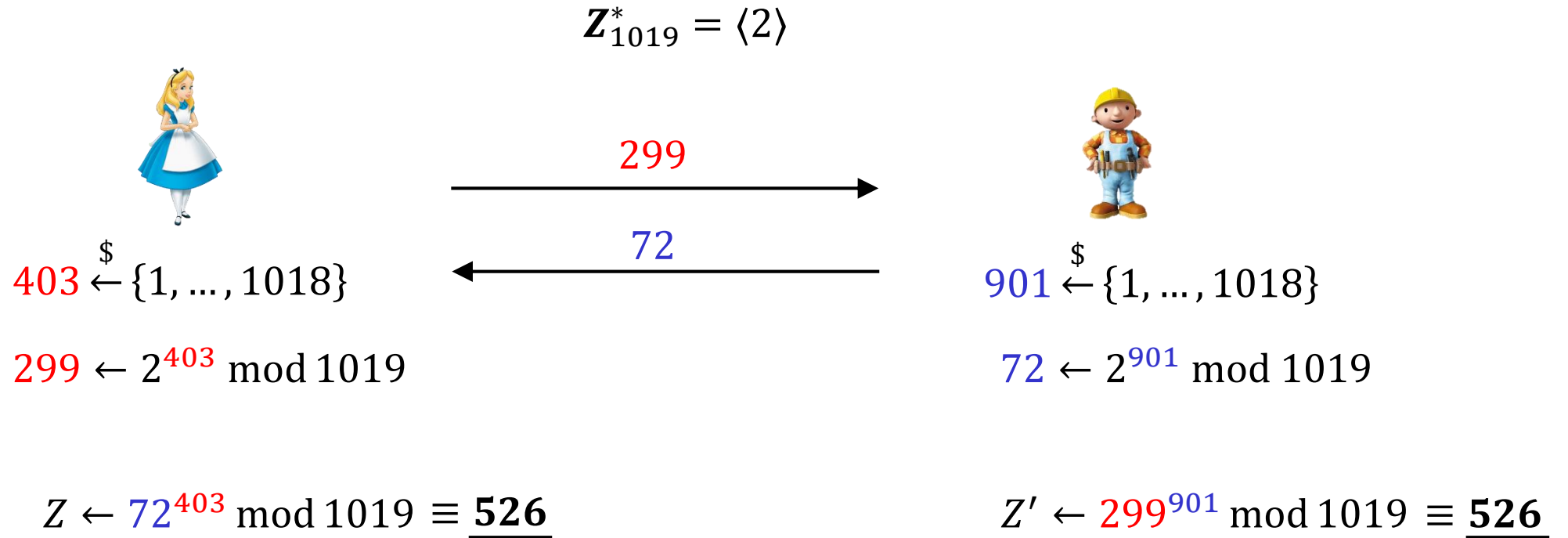


Diffie-Hellman

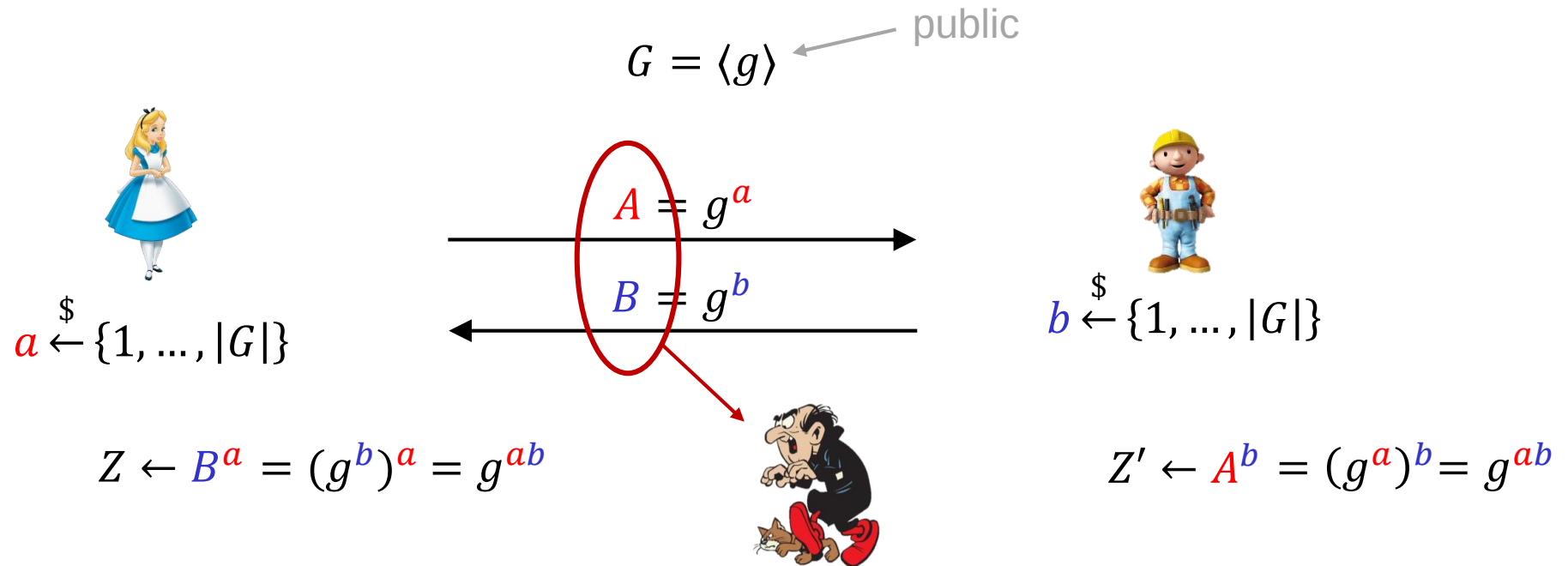


Claim: $Z = Z'$

Diffie-Hellman – example



Diffie-Hellman



Security:

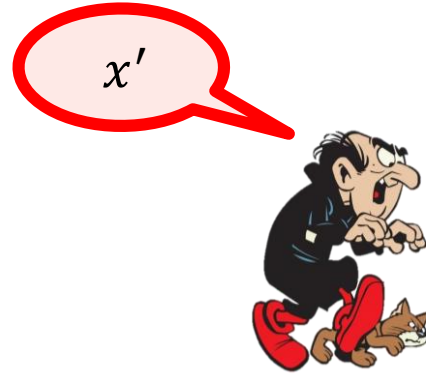
- Must be hard to compute $Z \leftarrow g^{ab}$ given g, A, B (DH assumption)
- Must be hard to find a (or b) given g, A, B (DLOG assumption)

Doesn't work: $A \circ B = g^a \circ g^b = g^{a+b} \neq g^{ab}$

Discrete logarithm (DLOG) problem

Exp_{G,g}^{dlog}(A)

1. $x \xleftarrow{\$} \{1, 2, \dots, |G|\}$
2. $X \leftarrow g^x$
3. $x' \leftarrow A(X)$
4. **return** $g^{x'} \stackrel{?}{=} X$



public

$$G = \langle g \rangle$$

Challenger

$$x \xleftarrow{\$} \{1, 2, \dots, |G|\}$$

$$X \leftarrow g^x$$

$$\xleftarrow{X}$$

Adversary wins if $g^{x'} = X$

In other words: $x' = \log_g X$

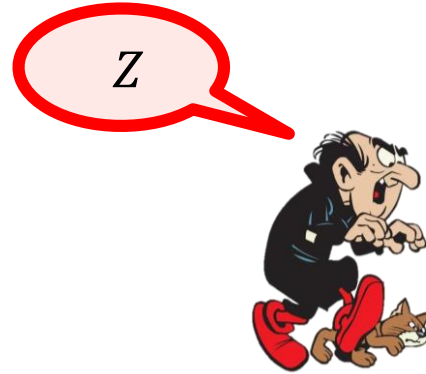
Definition: The **DLOG-advantage** of an adversary A is

$$\mathbf{Adv}_{G,g}^{\text{dlog}}(A) = \Pr \left[\mathbf{Exp}_{G,g}^{\text{dlog}}(A) \Rightarrow \text{true} \right]$$

Diffie-Hellman (DH) problem

$\text{Exp}_{G,g}^{\text{dh}}(A)$

1. $x, y \xleftarrow{\$} \{1, 2, \dots, |G|\}$
2. $X \leftarrow g^x$
3. $Y \leftarrow g^y$
4. $Z \leftarrow A(X, Y)$
5. **return** $Z \stackrel{?}{=} g^{xy}$



Adversary wins if $Z = g^{xy}$

public
 $G = \langle g \rangle$

$\xleftarrow{X, Y}$

Challenger

$x, y \xleftarrow{\$} \{1, 2, \dots, |G|\}$

$X \leftarrow g^x$

$Y \leftarrow g^y$

Definition: The **DH-advantage** of an adversary A is

$$\text{Adv}_{G,g}^{\text{dh}}(A) = \Pr[\text{Exp}_{G,g}^{\text{dh}}(A) \Rightarrow \text{true}]$$

DLOG vs. DH

Exp_{G,g}^{dlog}(A)

1. $x \xleftarrow{\$} \{1, 2, \dots, |G|\}$
2. $X \leftarrow g^x$
3. $x' \leftarrow A(X)$
4. **return** $g^{x'} \stackrel{?}{=} x$

Exp_{G,g}^{dh}(A)

1. $x, y \xleftarrow{\$} \{1, 2, \dots, |G|\}$
2. $X \leftarrow g^x$
3. $Y \leftarrow g^y$
4. $Z \leftarrow A(X, Y)$
5. **return** $Z \stackrel{?}{=} g^{xy}$

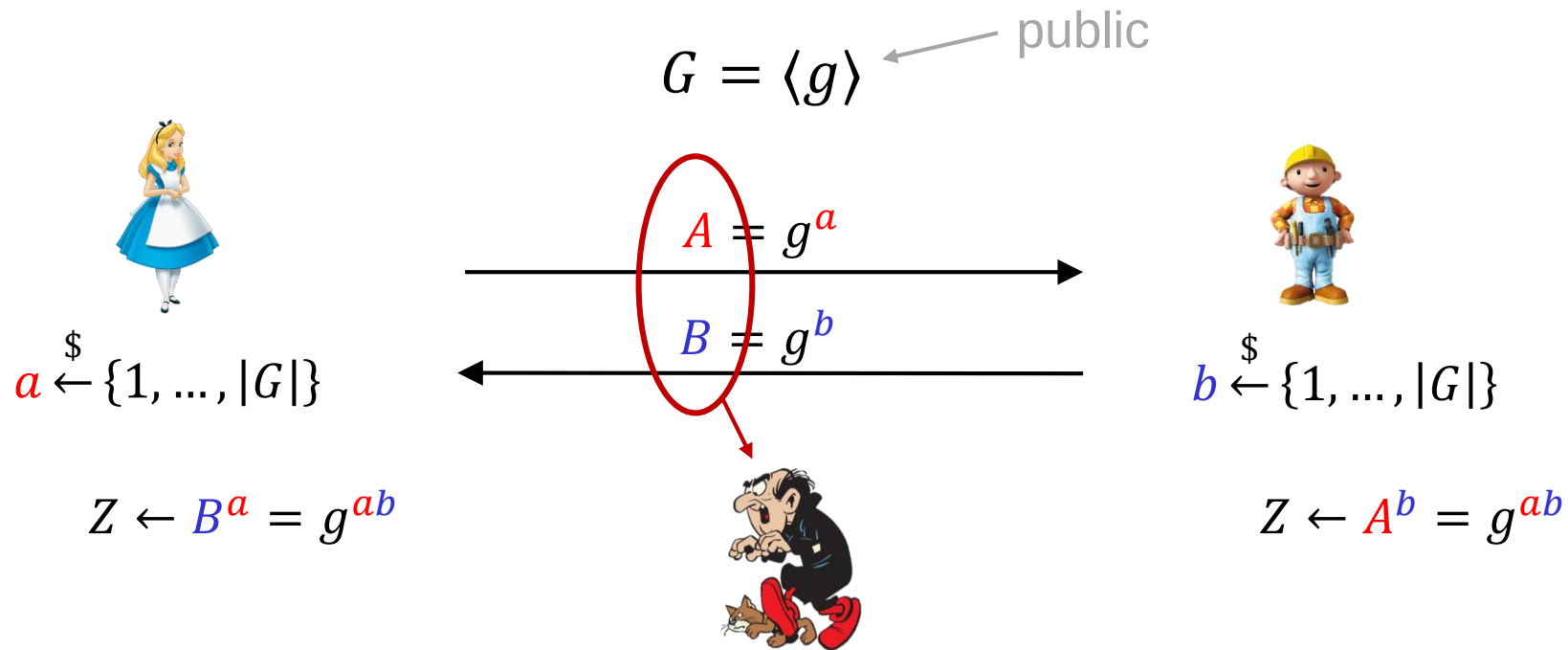
DLOG security $\stackrel{?}{\Rightarrow}$ DH security

DLOG security $\Leftarrow$ DH security



DLOG *insecurity* $\Rightarrow$ DH *insecurity*

Diffie-Hellman – security



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

- Must be hard to compute $Z \leftarrow g^{ab}$
- Must be hard to find a (or b)
- $G = (\mathbb{Z}_n, +)$
- $G = (\mathbb{Z}_p^*, \cdot)$

(DH assumption)

(DLOG assumption)

not secure

secure for $|\mathbb{Z}_p^*|$ large enough

(conjectured)

Algorithms for solving DLOG

- **Generic algorithms:** works for *all* (cyclic) groups
 - Brute-force
 1. Given g and $X \in G$
 2. for $i = 1, 2, \dots, |G|$ check if $g^i = X$
 - Are there better algorithms?
- **run time:** $\mathcal{O}(|G|) = (2^n)$, assuming $|G| \approx 2^n$
- **Group-specific algorithms:** exploits algebraic features of given group

Solving DLOG: the baby-step giant-step algorithm

Given: $X \leftarrow g^x$

$$n \leftarrow \lceil \sqrt{|G|} \rceil$$

$Y \leftarrow g^n$

Find: x

Look for i, j such that:

$$Xg^i = Y^j$$

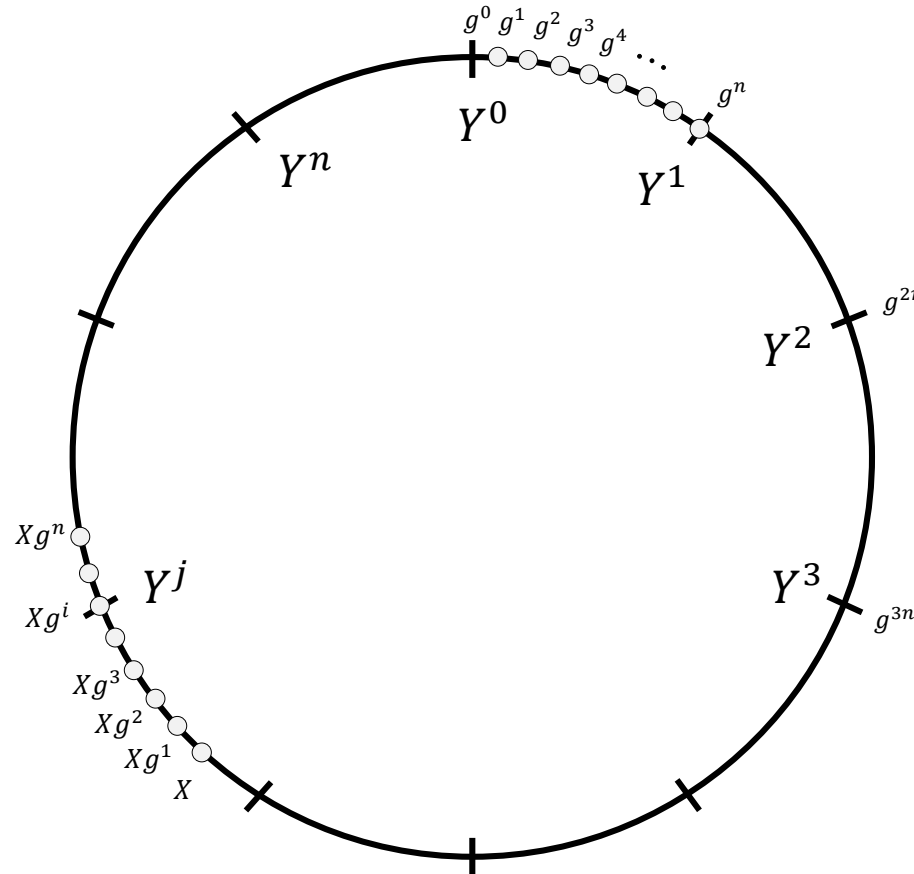
$$X = Y^j g^{-i}$$

$$X = g^{nj} g^{-i}$$

$$g^x = g^{nj} g^{-i}$$

$$g^x = g^{nj-i}$$

$$x = nj - i$$



$$\underbrace{O(\sqrt{|G|})}_{Y^0, Y^1, \dots, Y^n}$$

$$\underbrace{Xg^0, Xg^1, \dots, Xg^n}_{\tilde{O}(\sqrt{|G|})}$$

$$\text{Time + memory: } \tilde{O}(\sqrt{|G|})$$

Generic algorithms for solving DLOG

- Baby-step, giant-step: time $\mathcal{O}(\sqrt{|G|})$ memory $\mathcal{O}(\sqrt{|G|})$
- Pollard's rho: time $\mathcal{O}(\sqrt{|G|})$ memory $\mathcal{O}(1)$
- Pohlig-Hellman: time $\max_{p_i} \mathcal{O}(\sqrt{p_i})$ memory $\mathcal{O}(1)$ **One of the p_i must be large enough!**
- **Consequence:** for DLOG to be hard $\sqrt{|G|}$ must be large enough!
 - $|G| \approx 2^{128}$ only gives $\sqrt{2^{128}} = 2^{64}$ security
 - $|G| \approx 2^{256}$ only gives $\sqrt{2^{256}} = 2^{128}$ security
 - $|G| \approx 2^{512}$ only gives $\sqrt{2^{512}} = 2^{256}$ security
 - etc...
- **Nechaev'94 & Shoup'97:** Solving DLOG requires $\Omega(\sqrt{|G|})$ time in *generic* groups

Non-generic algorithms for DLOG

- Unfortunately, $(\mathbf{Z}_p^*, \cdot)$ is *not* a generic group!
- *Much* faster specific algorithms exist for solving DLOG in $\mathbf{Z}_p^*$
 - 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: $|\mathbf{Z}_p^*| \approx 2^{795}$ using GNFS (Heninger et al. '19)
 - Previous records: https://en.wikipedia.org/wiki/Discrete_logarithm_records
- $|\mathbf{Z}_p^*| \geq 2^{2048}$ typically required as a minimum today

Diffie-Hellman in $(\mathbb{Z}_p^*, \cdot)$ – what is large enough?

$p =$ 17125458317614137930196041979257577826408832324037508573393292981642667139747621778802438775238728592968344613589379932348475613503476932163166973813218698343816463289144185362912602522540494983090531497232965829
53652450726984882565831142029933592229570974326750832252596677395039491925757684203877163274204414247105350985012360588381585716266691777519349615737265619555830572700989127600651400040936587721817138831992389630
93777917625906143118496429613802248519404604217104493688927525974870395873936387909672274883295377481008150475878590270591798350563488168080923804611822387520198054002990623911454389104774092183 $\approx 2^{2048}$



$A = 2$

$\text{mod } p$



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

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

$B = 2$

$\text{mod } p$

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

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

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

$\times$

$Z \leftarrow 2$

$\text{mod } p$

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

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

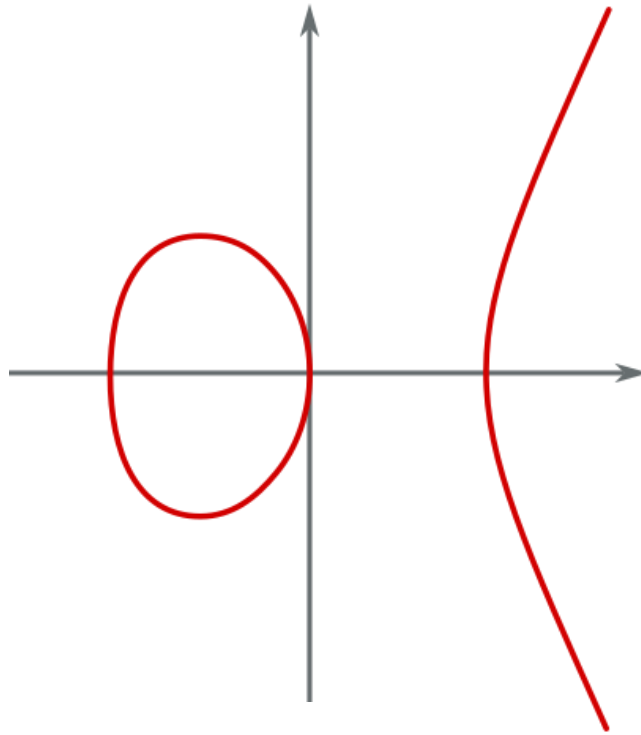


Z_p^*

Groups
where GNFS
doesn't work

Better alternatives to Z_p^* ?

Elliptic curves



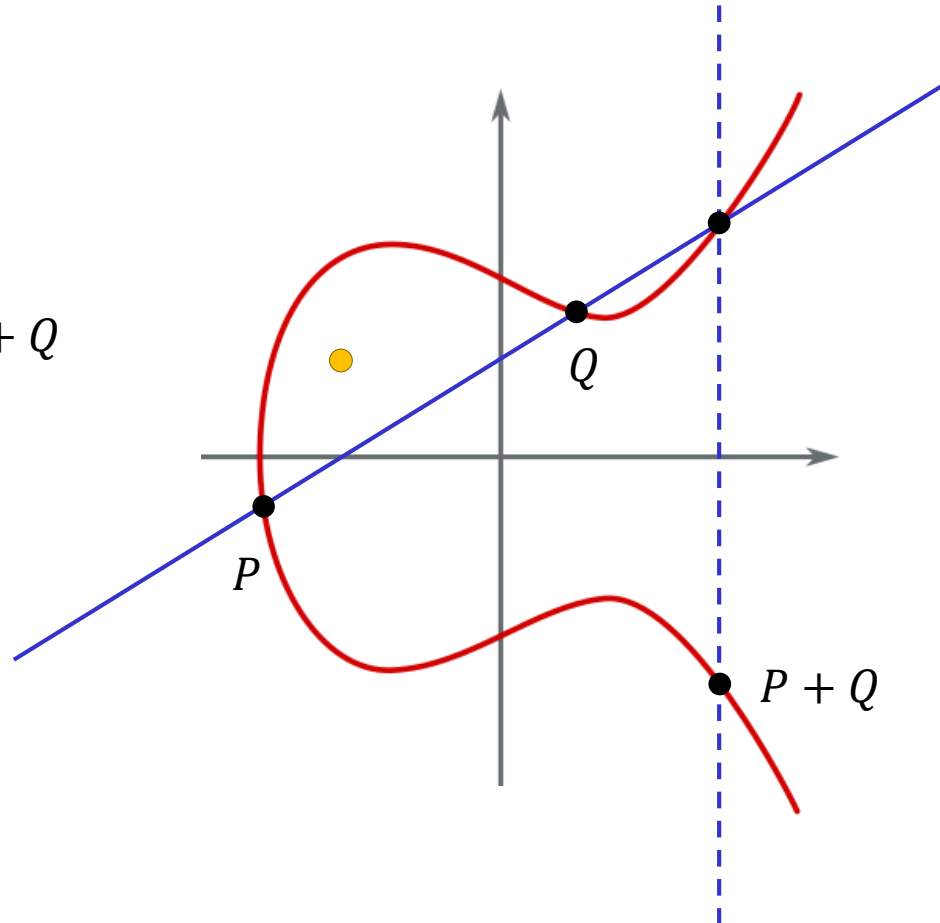
$$y^2 = x^3 - 9x - 1 \quad x, y \in \mathbf{R}$$

Elliptic curves

$$P = (x, y)$$

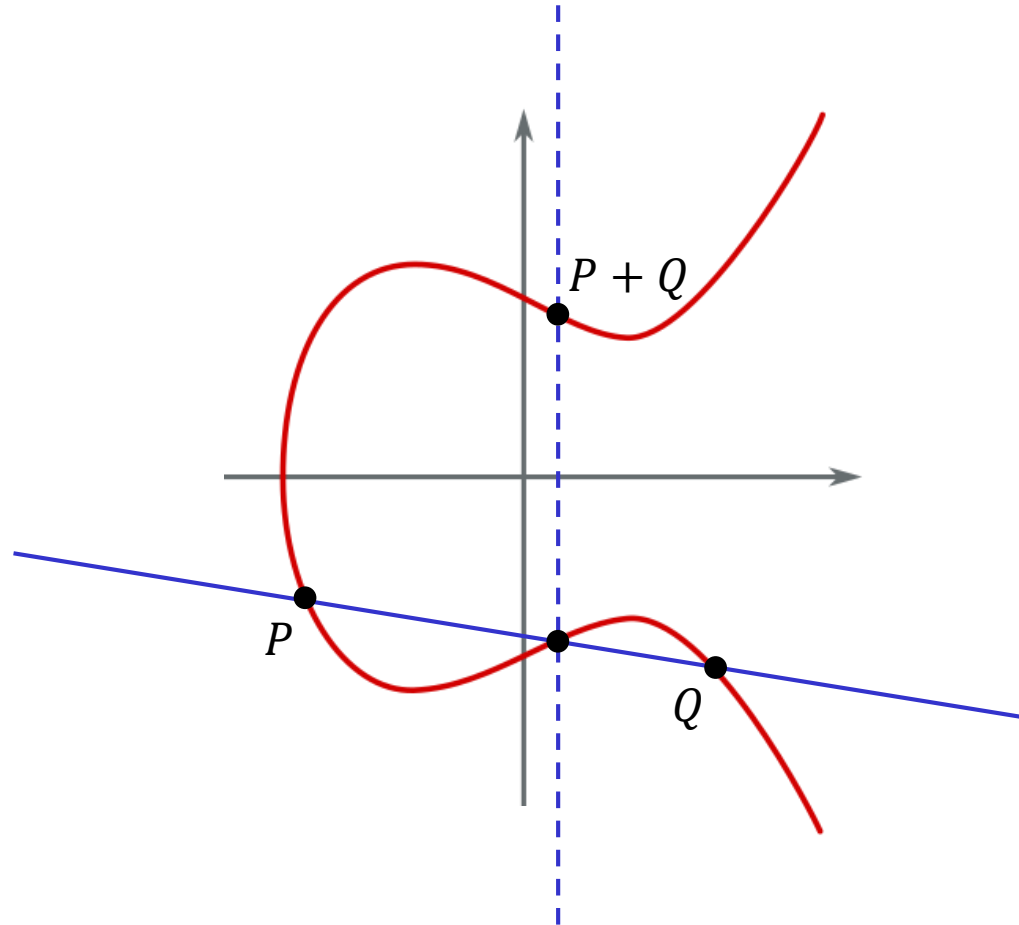
$$Q = (x', y')$$

$$(x + x', y + y') \neq P + Q$$



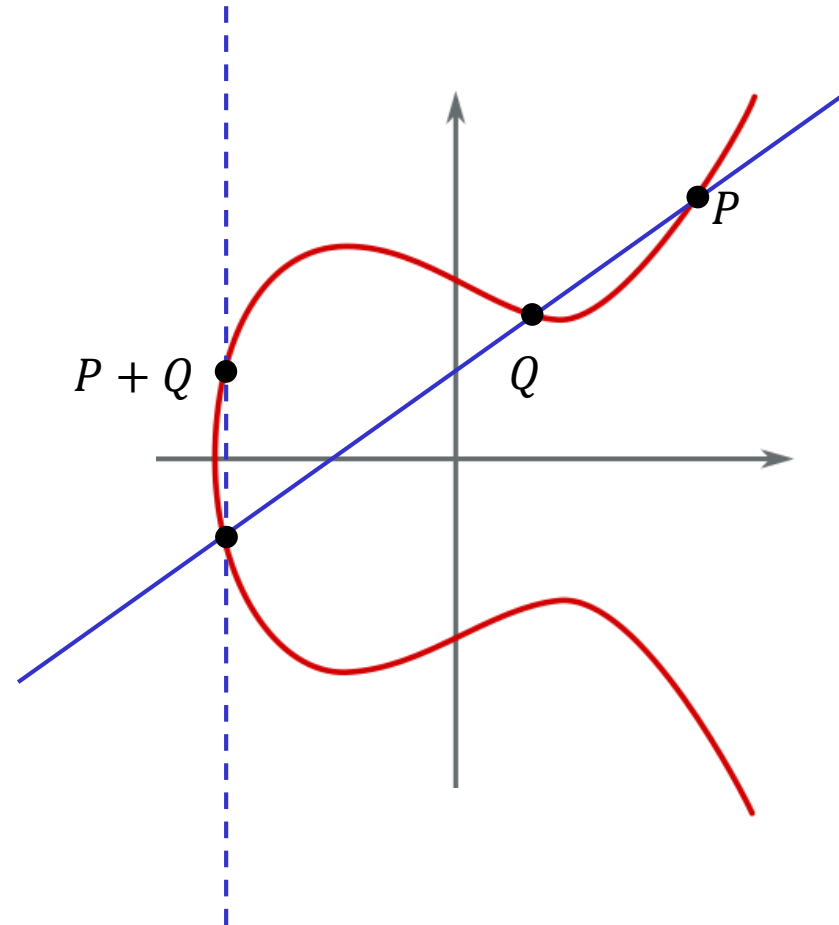
$$y^2 = x^3 - 2x + 3 \quad x, y \in \mathbf{R}$$

Elliptic curves



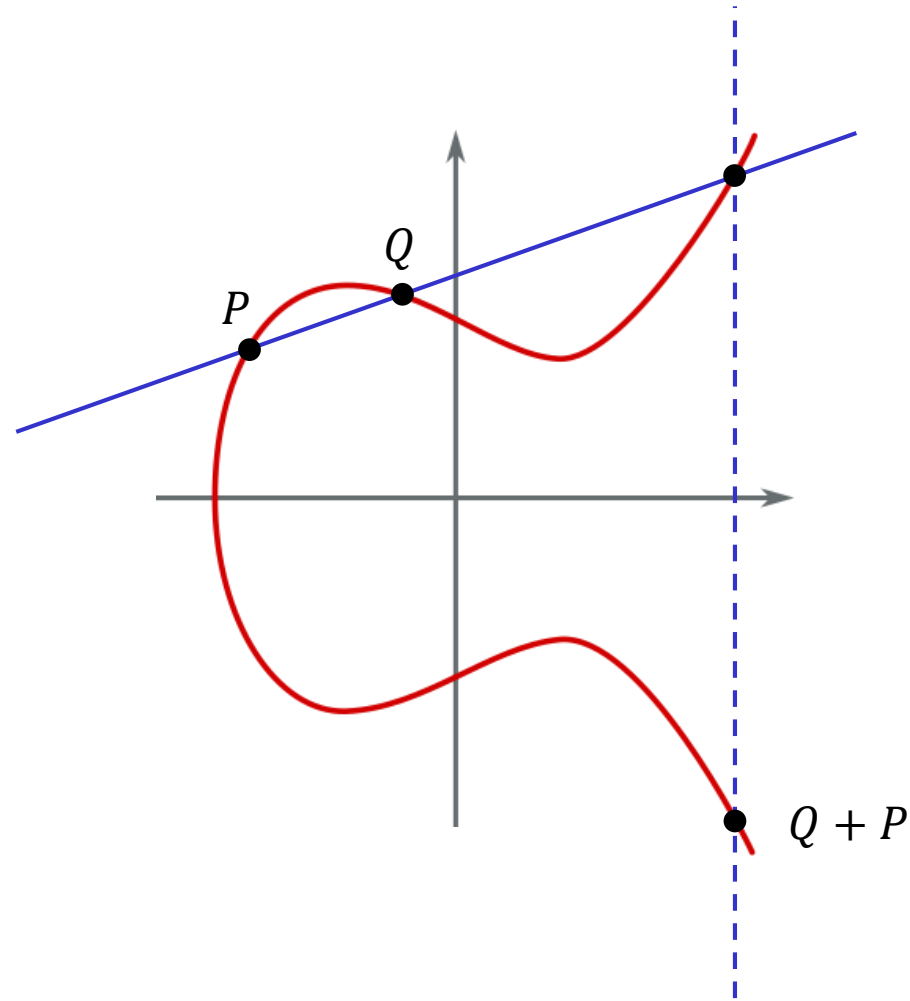
$$y^2 = x^3 - 2x + 3 \quad x, y \in \mathbf{R}$$

Elliptic curves



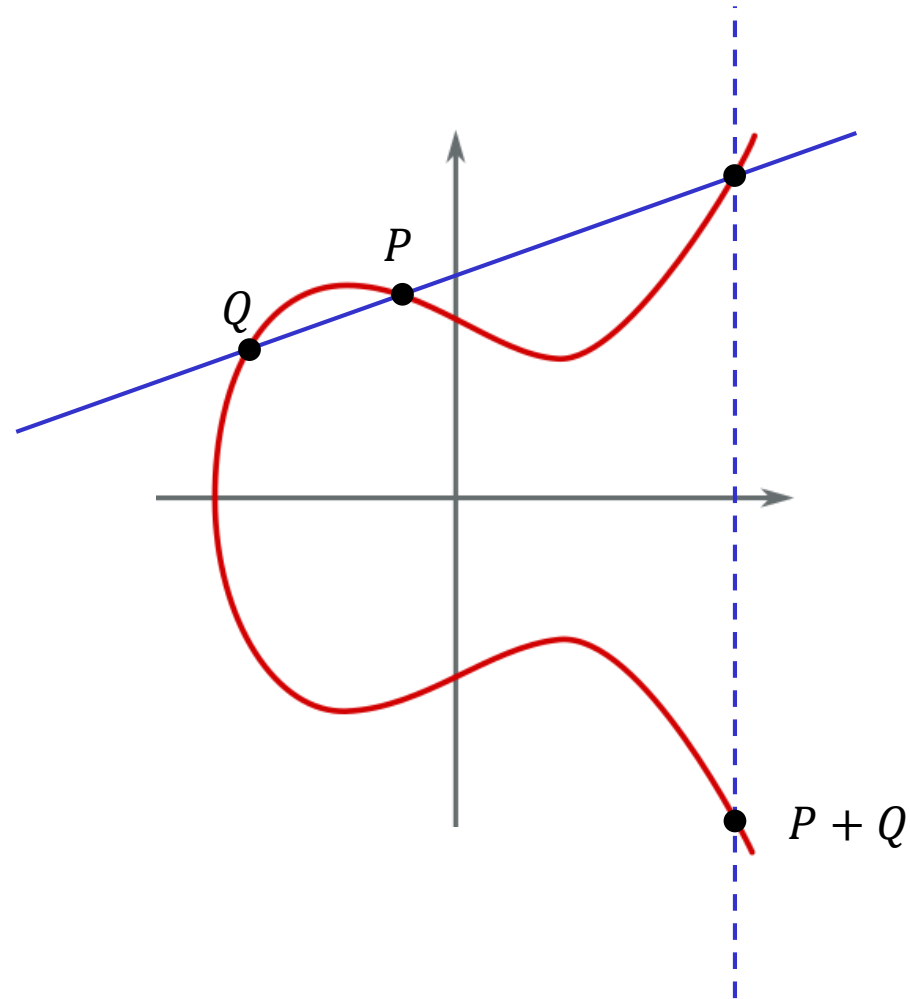
$$y^2 = x^3 - 2x + 3 \quad x, y \in \mathbf{R}$$

Elliptic curves



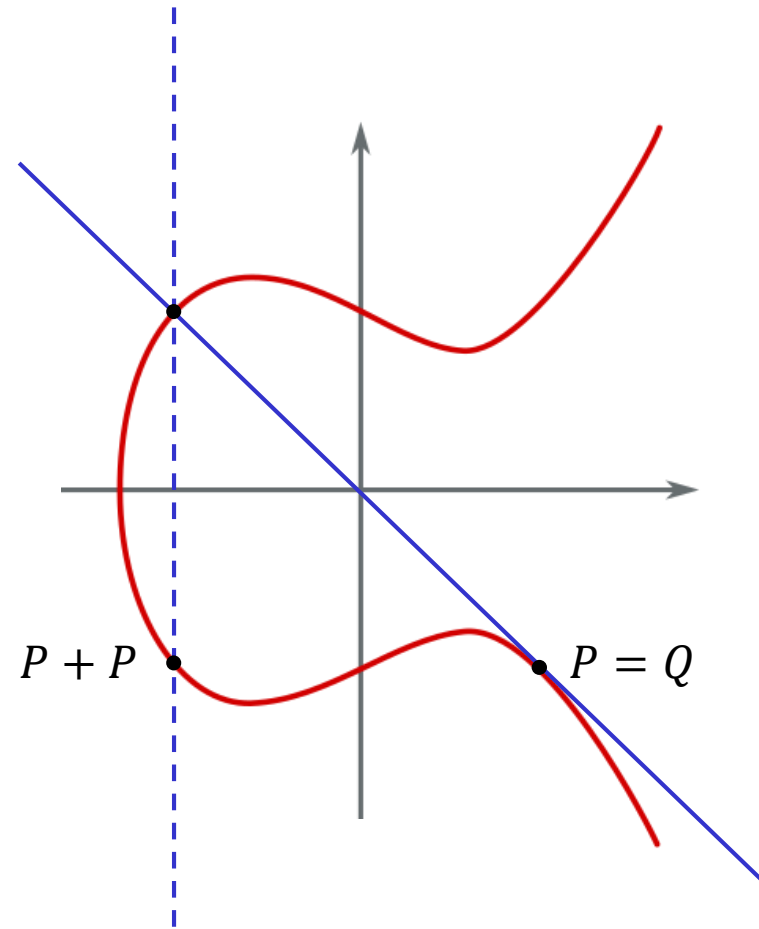
$$y^2 = x^3 - 2x + 3 \quad x, y \in \mathbb{R}$$

Elliptic curves



$$y^2 = x^3 - 2x + 3 \quad x, y \in \mathbf{R}$$

Elliptic curves

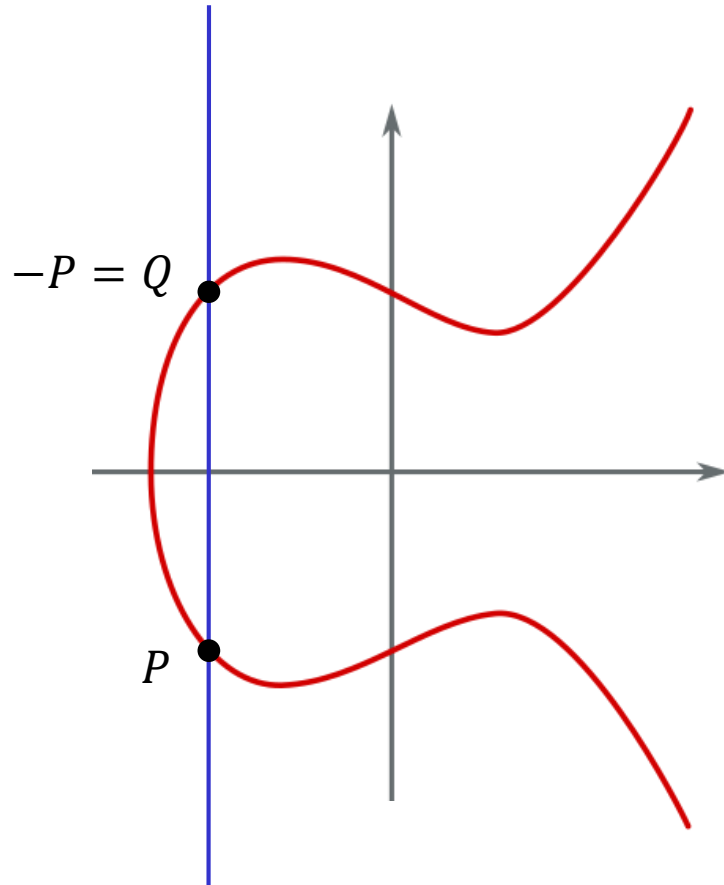


$$y^2 = x^3 - 2x + 3 \quad x, y \in \mathbf{R}$$

Elliptic curves

$$P + Q \stackrel{\text{def}}{=} \mathcal{O}$$

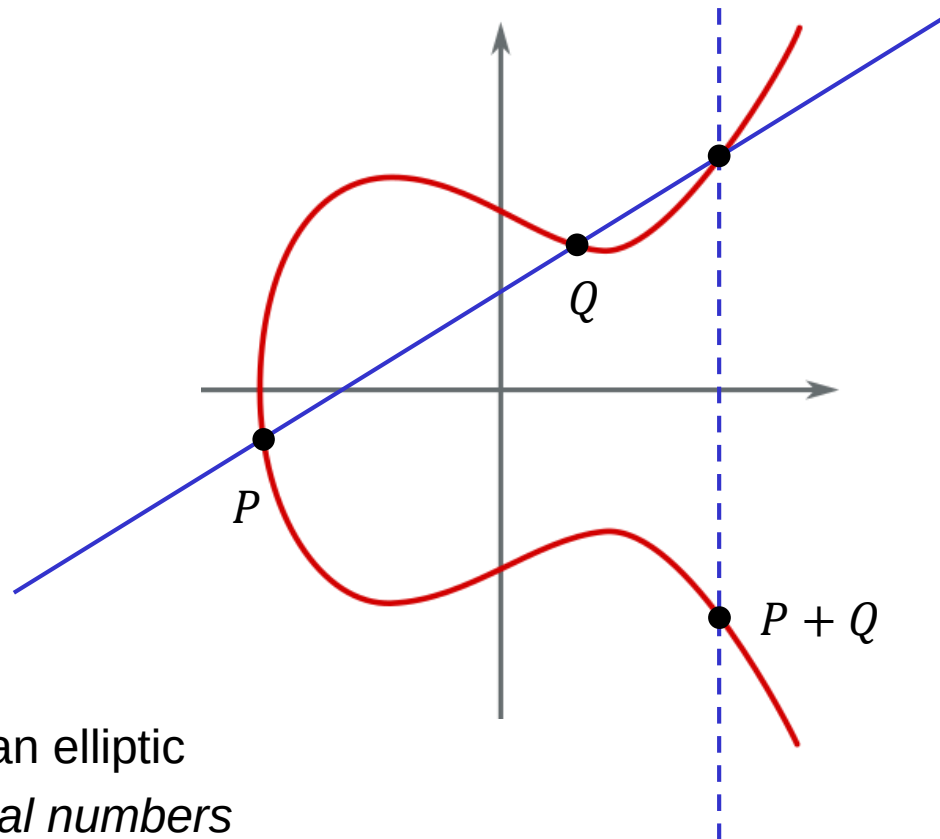
Identity element



$$y^2 = x^3 - 2x + 3 \quad x, y \in \mathbf{R}$$

Elliptic curves

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



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

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

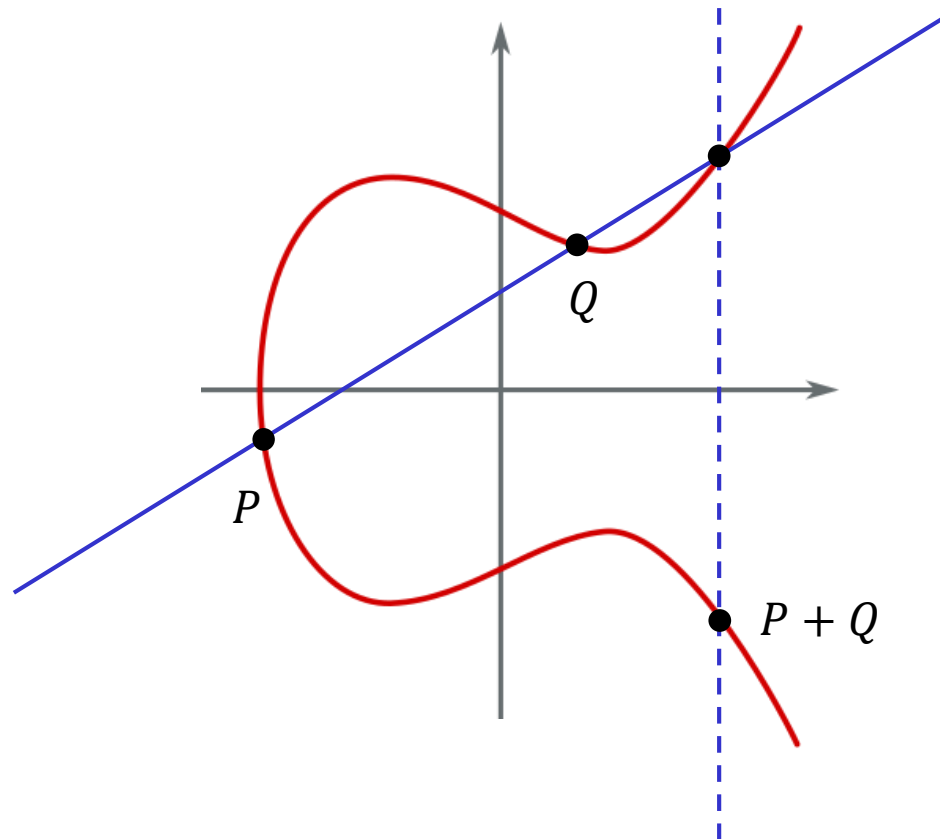
$$a, b, x, y \in \mathbf{R}$$

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

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

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

Elliptic curves over $\mathbb{Z}_p$



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

$$a, b, x, y \in \mathbb{Z}_p$$

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

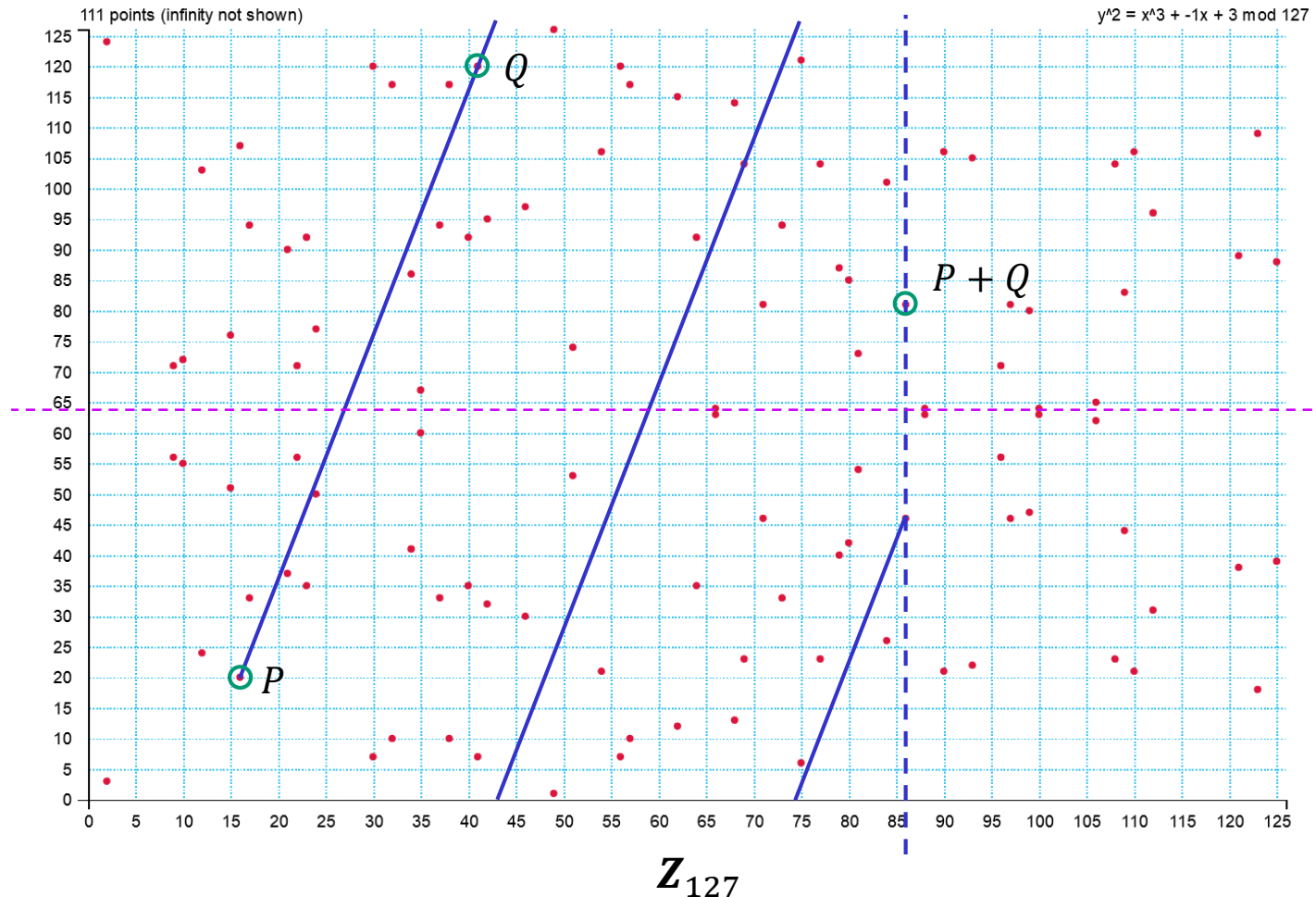
$$x_3 = \frac{(x_1x_2 - 2a)x_1x_2 - 4b(x_1 + x_2) + a^2}{(x_1x_2 + a)(x_1 + x_2) + 2y_1y_2 + 2b}$$

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

Elliptic curves over $\mathbb{Z}_p$

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

Notation: $(E(\mathbb{F}_p), +)$ = an elliptic curve group defined over $\mathbb{F}_p$



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

$$a, b, x, y \in \mathbb{Z}_p$$

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

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

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

$$(\mathbb{Z}_p, +) \quad (\mathbb{Z}_p^*, \cdot)$$

$$(\mathbb{F}_p, +, \cdot)$$

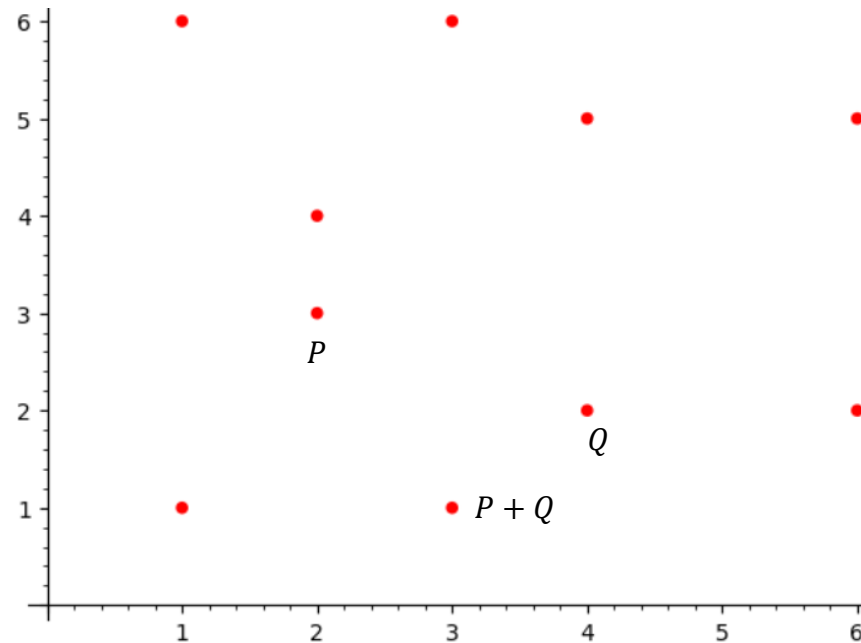
Finite field

Still valid!

Elliptic curves – example

$$y^2 = x^3 + x - 1$$

$$(E(\mathbf{Z}_7), +)$$



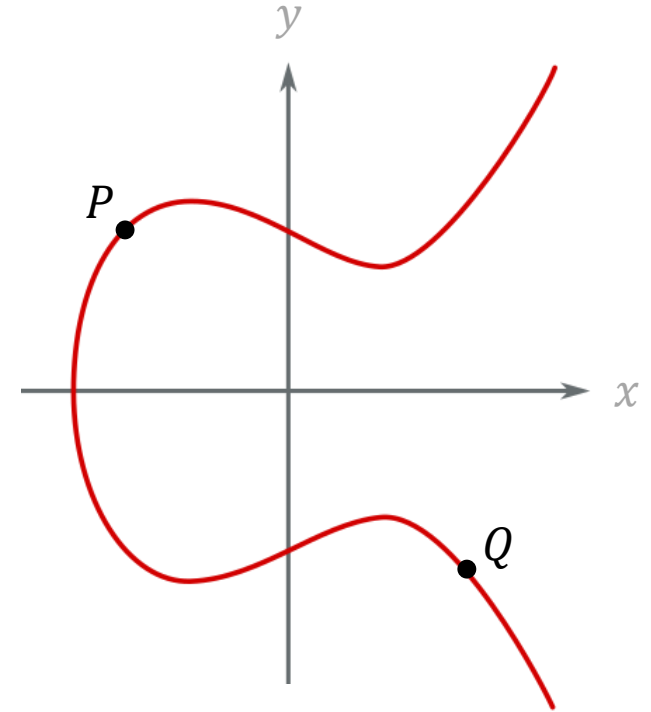
$$(E(\mathbf{Z}_7), +) = \{\mathcal{O}, (1,1), (1,6), (2,3), (2,4), (3,1), (3,6), (4,2), (4,5), (6,2), (6,5)\}$$

Elliptic curves summary

- $P = (x, y)$ point on elliptic curve
- x and y are elements in $\mathbf{Z}_p$
- 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 for **secret** n , find n



$E(\mathbf{Z}_p)$ – properties

- $(\mathbf{Z}_p^*, \cdot)$ is *not* a generic group
 - Specialized attacks (GNFS) exploit algebraic structure $\Rightarrow$ parameters must be bigger to compensate
 - $|\mathbf{Z}_p^*| \geq 2^{2048}$ required for security today
 - Bigger parameters $\Rightarrow$ slower systems
- Currently no attacks manage to exploit the algebraic structure of $(E(\mathbf{Z}_p), +)$
 - Best-known attacks are *generic attacks*:
 - **Nechaev '94 & Shoup '97**: *Generic* algorithms for solving DLOG require time $\Omega(\sqrt{|G|})$
 - Consequently: elliptic curve crypto can use *much* smaller parameters
 - $|E(\mathbf{Z}_p)| = 2^{256}, 2^{384}, 2^{512}$ common in practice $\Rightarrow$ much faster than $\mathbf{Z}_p^*$ -based crypto

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

$p =$

171254583176141379301960419792575778264088323240375085733932929816426671397476217788024387752387285929683446135893799323484756135034769321631669738132186983438164632891441853629126025225404949830905314972329658295365245072698488256583114202993359222957097432675083225259667739503949192575768420387716327420441424710535098501236058838158571626669177751934961573726561955583057270098912760065140004093658772181713883199238963093777917625906143118496429613802248519404604217104493688927525974870395873936387909672274883295377481008150475878590270591798350563488168080923804611822387520198054002990623911454389104774092183

$\approx 2^{2048}$



$$A = 2$$

$$\text{mod } p$$



$$B = 2$$

$$\text{mod } p$$

$\$$

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

$\$$

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

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

665680077810100734070476416898417408020670352441378967
997224101900193957211854827569676933273935982997835638
067380762809024311842715496660113976613131478051784487
322446809608196031803691263486170912693625523249742383
264476433978409434004770395167011614217279552198029936
108023398643999746539201834933168220864789880837484536
563067997761270826642223539437541791058416731683176995
365899867720283590420932417377927106884773199080517477
615608675490982255098980545292032908358093913531433702
04942904741307152555630729954144551438972122256380833
743627991333762777307118317040885305789910094293548481
5501309561942770687524

665680077810100734070476416898417408020670352441378967
997224101900193957211854827569676933273935982997835638
067380762809024311842715496660113976613131478051784487
322446809608196031803691263486170912693625523249742383
264476433978409434004770395167011614217279552198029936
108023398643999746539201834933168220864789880837484536
563067997761270826642223539437541791058416731683176995
365899867720283590420932417377927106884773199080517477
615608675490982255098980545292032908358093913531433702
04942904741307152555630729954144551438972122256380833
743627991333762777307118317040885305789910094293548481
5501309561942770687524

$$Z \leftarrow 2$$

$\times$

$$\text{mod } p$$

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

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



$$A = \begin{matrix} 890802873957740768864044652762626807642 \\ 97347818898641719959470631357066012729 \end{matrix} \cdot Q$$



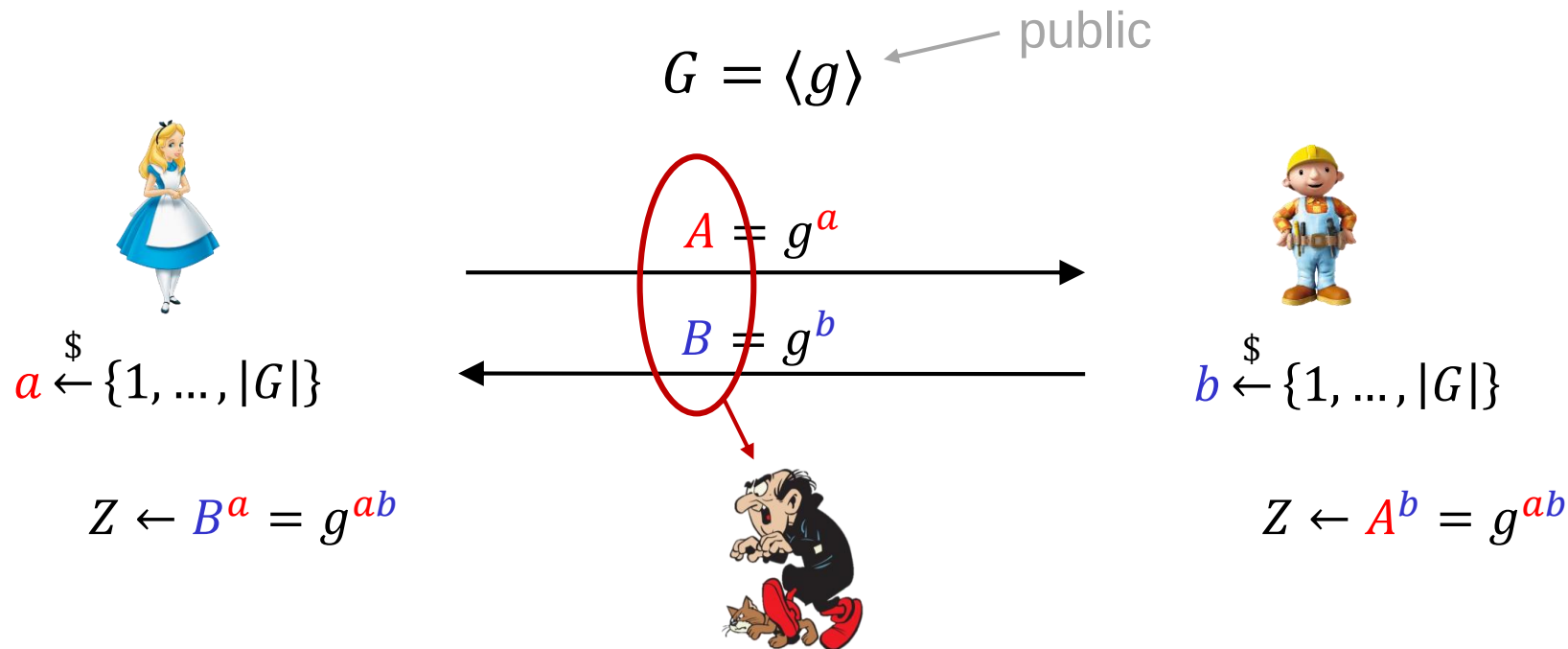
$$\begin{matrix} 890802873957740768864044652762626807642 \\ 97347818898641719959470631357066012729 \end{matrix} \stackrel{\$}{\leftarrow} \{1 \dots p\}$$

$$\begin{matrix} 477777099444069171196101629885784192859 \\ 74852857762532427086507801748071058272 \end{matrix} \stackrel{\$}{\leftarrow} \{1 \dots p\}$$

$$B = \begin{matrix} 477777099444069171196101629885784192859 \\ 74852857762532427086507801748071058272 \end{matrix} \cdot Q$$

$$Z \leftarrow \begin{matrix} 890802873957740768864044652762626807642 \\ 97347818898641719959470631357066012729 \end{matrix} \times \begin{matrix} 477777099444069171196101629885784192859 \\ 74852857762532427086507801748071058272 \end{matrix} \cdot Q$$

Diffie-Hellman – security



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

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

(DH assumption)

(DLOG assumption)

not secure

- $G = (\mathbf{Z}_p, +)$

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

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

secure for $|\mathbf{Z}_p^*|$ large enough

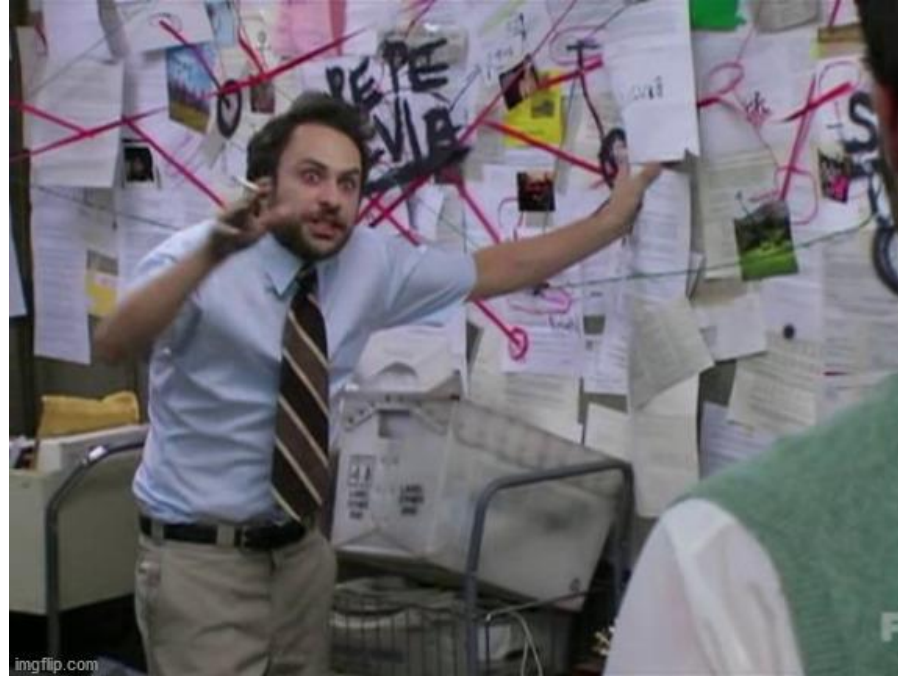
(conjectured)

secure for $|E(\mathbf{F}_p)|$ large enough

(conjectured)

Cryptographic groups in practice

- $(\mathbf{Z}_p^*, \cdot)$ groups:
 - **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)
 - all p 's are **safe primes** (i.e., of the form $p = 2q + 1$ where q is prime)
- $(E(\mathbf{Z}_p^*), +)$ groups
 - 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)



Now for something completely different...

Juniper Networks

- Juniper Networks: big manufacturer of network equipment (routers, VPNs, firewalls, etc.)
 - Major customers: telcos, banks, US DoD
- 2015:

IMPORTANT JUNIPER SECURITY ANNOUNCEMENT

During a recent internal code review, Juniper discovered **unauthorized code in ScreenOS** that could allow a knowledgeable attacker to gain administrative access to NetScreen® devices and to decrypt VPN connections.

- Hackers obtained access to source code repository
- Only made one change:

```
--- Qx = 2c55e5e45edf713dc43475effe8813a60326a64d9ba3d2e39cb639b0f3b0ad10  
+++ Qx = 9585320eeaf81044f20d55030a035b11bece81c785e6c933e4a8a131f6578107
```

JUNIPER[®]
NETWORKS



PRNG standardization

NIST Special Publication 800-90

Recommendation for Random Number Generation Using Deterministic Random Bit Generators

Elaine Barker
John Kelsey

Computer Security Division
Information Technology Laboratory

COMPUTER SECURITY

June 2006



U.S. Department of Commerce
Carlos M. Gutierrez, Secretary

Technology Administration
Robert Cresanti, Under Secretary of Commerce for Technology

National Institute of Standards and Technology
William Jeffrey, Director

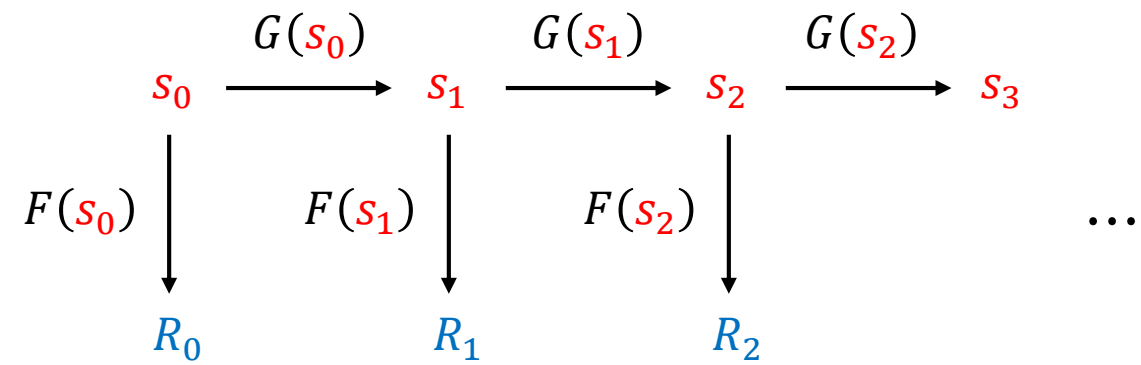
NIST National Institute of Standards and Technology • Technology Administration • U.S. Department of Commerce

NIST SP 800-90

June 2006

8.7.2	Additional Input.....	21
8.8	Prediction Resistance and Backtracking Resistance	21
9	DRBG Mechanism Functions	24
9.1	Instantiating a DRBG	24
9.2	Reseeding a DRBG Instantiation.....	27
9.3	Generating Pseudorandom Bits Using a DRBG	29
9.3.1	The Generate Function.....	29
9.3.2	Reseeding at the End of the Seedlife.....	32
9.3.3	Handling Prediction Resistance Requests.....	33
9.4	Removing a DRBG Instantiation.....	33
10	DRBG Algorithm Specifications	35
10.1	DRBG Mechanisms Based on Hash Functions.....	35
10.1.1	Hash_DRBG	36
10.1.1.1	Hash_DRBG Internal State.....	36
10.1.1.2	Instantiation of Hash_DRBG.....	37
10.1.1.3	Reseeding a Hash_DRBG Instantiation.....	38
10.1.1.4	Generating Pseudorandom Bits Using Hash_DRBG	39
10.1.2	HMAC_DRBG.....	41
10.1.2.1	HMAC_DRBG Internal State.....	41
10.1.2.2	The Update Function (Update)	42
10.1.2.3	Instantiation of HMAC_DRBG.....	43
10.1.2.4	Reseeding an HMAC_DRBG Instantiation	43
10.1.2.5	Generating Pseudorandom Bits Using HMAC_DRBG	44
10.2	DRBG Mechanisms Based on Block Ciphers	46
10.2.1	CTR_DRBG	46
10.2.1.1	CTR_DRBG Internal State	49
10.2.1.2	The Update Function (Update)	49
10.2.1.3	Instantiation of CTR_DRBG	50
10.2.1.4	Reseeding a CTR_DRBG Instantiation	52
10.2.1.5	Generating Pseudorandom Bits Using CTR_DRBG.....	54
10.3	DRBG Mechanisms Based on Number Theoretic Problems	58
10.3.1	Dual Elliptic Curve Deterministic RBG (Dual_EC_DRBG)	58

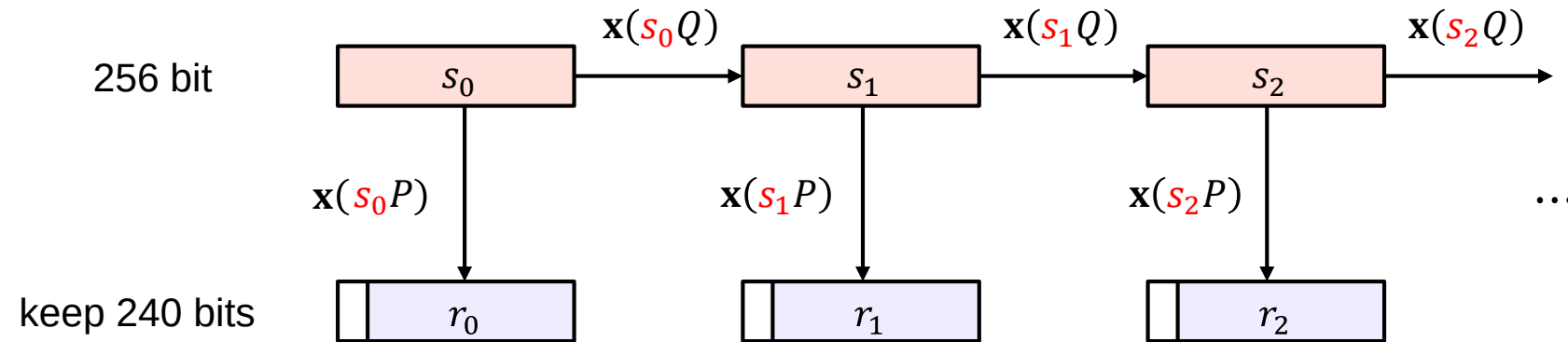
PRNG



Dual EC DRBG

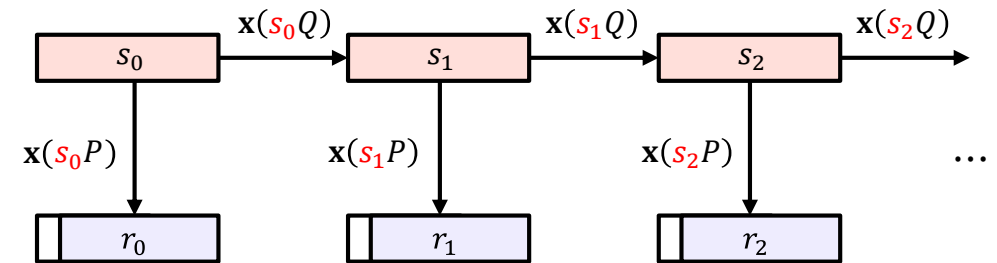
P, Q : **public** points on an elliptic curve

$x(\cdot)$: x-coordinate of elliptic curve point (256 bits)



Dual EC DRBG – something's fishy

- Dual EC is *slow*
 - Orders of magnitude slower than HMAC/AES-CTR based alternatives
- **2006** – Kristian Gjøsteen: Dual EC is *not* a good PRNG
 - Can distinguish output from random with $\text{Adv}_{\text{DualEC}}^{\text{prg}}(KG) \approx 0.0011$
 - Slightly improved by Schoenmakers and Sidorenko
- **2007** – Shumow and Ferguson: Dual EC can be *backdoored*
 - What if $Q = kP$ for a secret k only you know?

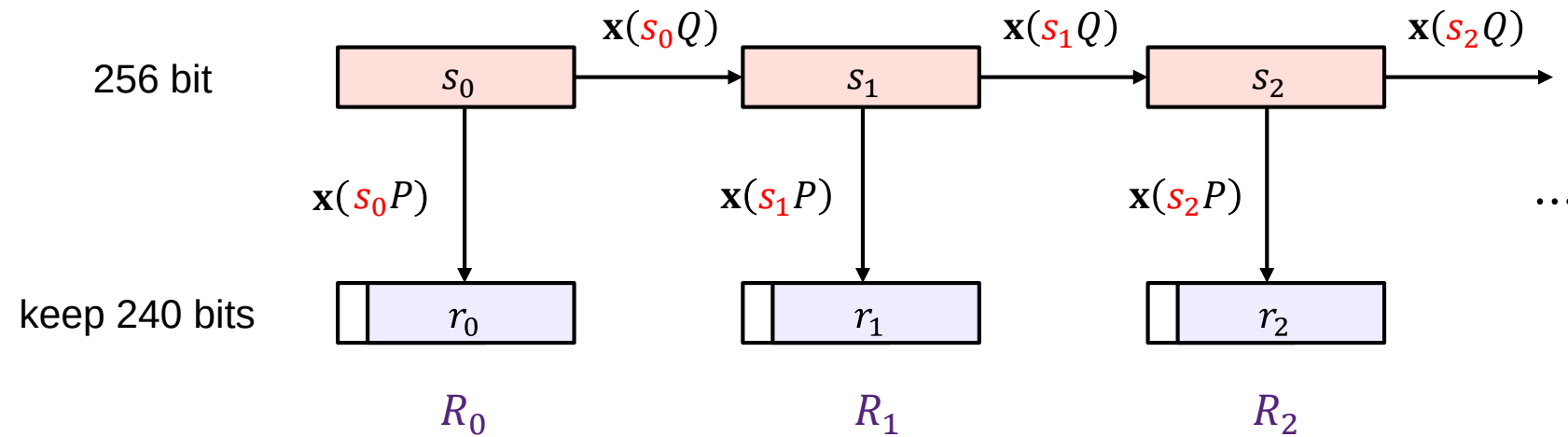


Dual EC DRBG

P, Q : **public** points on an elliptic curve

$$Q = kP$$

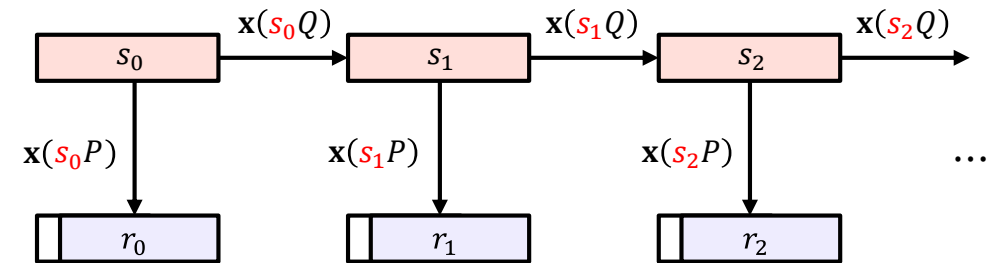
$\mathbf{x}(\cdot)$: x-coordinate (256 bits)



$$kR_0 = k \cdot (s_0 P) = s_0 \cdot (kP) = s_0 Q \xrightarrow{\mathbf{x}(\cdot)} s_1$$

Dual EC DRBG – something's fishy

- Dual EC is *slow*
 - Orders of magnitude slower than HMAC/AES-CTR based alternatives
- **2006** – Kristian Gjøsteen: Dual EC is *not* a good PRNG
 - Can distinguish output from random with $\text{Adv}_{\text{DualEC}}^{\text{prg}}(KG) \approx 0.0011$
 - Slightly improved by Schoenmakers and Sidorenko
- **2007** – Shumow and Ferguson: Dual EC can be *backdoored*
 - What if $Q = kP$ for a secret k only you know?
 - If you know full s_0P , compute $k(s_0P) = s_0(kP) = s_0Q \xrightarrow{x(\cdot)} s_1$
 - Because of truncation: need to guess top 16 bits ($\approx 2^{16}$ additional work)



Where does Q (and P) come from?

From: John Kelsey [mailto:john.kelsey@nist.gov]
Sent: Wednesday, October 27, 2004 11:17 AM
To: Don Johnson
Subject: Minding our Ps and Qs in Dual_EC

Do you know where Q comes from in Dual_EC_DRBG?

Thanks,
-Joh

Subject: RE: Minding our Ps and Qs in Dual_EC
From: "Don Johnson" <DJohnson@cygnacom.com>
Date: Wed, October 27, 2004 11:42 am
To: "John Kelsey" <john.kelsey@nist.gov>
John,

$P=G$.

Q is (in essence) the public key for some random private key.

It could also be generated like a(nother) canonical G , but NSA kyboshed this idea, and I was not allowed to publicly discuss it, just in case you may think of going there.

Don B. Johnson

NIST SP 800-90

June 2006

Appendix A: (Normative) Application-Specific Constants

A.1 Constants for the Dual_EC_DRBG

The **Dual_EC_DRBG** requires the specifications of an elliptic curve and two points on the elliptic curve. One of the following NIST approved curves with associated points **shall** be used in applications requiring certification under FIPS 140-2. More details about these curves may be found in FIPS PUB 186-3, the Digital Signature Standard.

Each of following curves is given by the equation:

$$y^2 = x^3 - 3x + b \pmod{p}$$

Notation:

p - Order of the field F_p , given in decimal

r - order of the Elliptic Curve Group, in decimal. Note that r is used here for consistency with FIPS 186-3 but is referred to as n in the description of the **Dual_EC_DRBG**.

a - (-3) in the above equation

b - coefficient above

The x and y coordinates of the base point, i.e., generator G , are the same as for the point P .

A.1.1 Curve P-256

p = 11579208921035624876269744694940757353008614\3415290314195533631308867097853951

r = 11579208921035624876269744694940757352999695\5224135760342422259061068512044369

b = 5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b

P_x = 6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945 d898c296

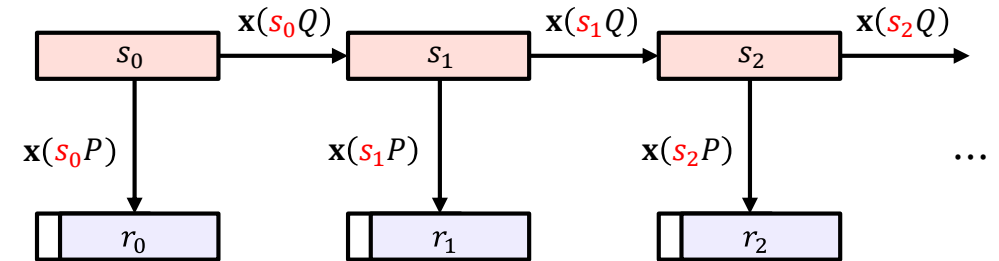
P_y = 4fe342e2 fe1a7f9b 8ee7eb4a 7c0f9e16 2bce3357 6b315ece cbb64068 37bf51f5

Q_x = c97445f4 5cdef9f0 d3e05e1e 585fc297 235b82b5 be8ff3ef ca67c598 52018192

Q_y = b28ef557 ba31dfcb dd21ac46 e2a91e3c 304f44cb 87058ada 2cb81515 1e610046

Dual EC DRBG – something's fishy

- Dual EC is *slow*
 - Orders of magnitude slower than HMAC/AES-CTR based alternatives
- **2006** – Kristian Gjøsteen: Dual EC is *not* a good PRNG
 - Can distinguish output from random with $\text{Adv}_{\text{DualEC}}^{\text{prg}}(KG) \approx 0.0011$
 - Slightly improved by Schoenmakers and Sidorenko
- **2007** – Shumow and Ferguson: Dual EC can be *backdoored*
 - What if $Q = kP$ for a secret k only you know?
 - If you know full s_0P , compute $k(s_0P) = s_0(kP) = s_0Q \xrightarrow{x(\cdot)} s_1$
 - Because of truncation: need to guess top 16 bits ($\approx 2^{16}$ additional work)
- **2007** – NIST adds appendix to standard on how to create P and Q yourself
 - Continues to recommend existing P and Q
- Most cryptographers: who cares? No one is going to use Dual EC anyway...
- **2013** – Edward Snowden leak: a project called Bullrun exists within the NSA
 - Purpose: "Insert vulnerabilities into commercial encryption systems, IT systems, networks, and endpoint communications devices used by targets."
 - Turns out Juniper Networks made Dual EC their PRNG in ScreenOS from 2008!



Juniper PRNG

ScreenOS does make use of the Dual_EC_DRBG standard, but is designed to not use Dual_EC_DRBG as its primary random number generator. ScreenOS uses it in a way that should not be vulnerable to the possible issue that has been brought to light. Instead of using the NIST recommended curve points it uses self-generated basis points and then takes the output as an input to FIPS/ANSI X.9.31 PRNG, which is the random number generator used in ScreenOS cryptographic operations.

Juniper Knowledge Base Article KB28205

Juniper PRNG

ScreenOS v.6.1

```
1. char seed[8] // dual_ec output
2. char block[8]; // X9.31 temporary output
3.
4. unsigned int calls_since_reseed;
5.
6. void prng_reseed(void) {
7.     unsigned int *output;
8.     dual_ec_generate(output, 32);
9.     memcpy(seed, output, 32);
10.    index = 32;
11. }
12.
13. void prng_generate(char *output) {
14.     unsigned int index = 0;
15.     calls_since_reseed++;
16.
17.     if (calls_since_reseed > 10 000)
18.         prng_reseed();
19.
20.     for (; index < 20; index += 8) {
21.         ...
22.         x9_31_generate_block(seed, block);
23.         ...
24.         memcpy(&output[index], block, 8);
25.     }
26. }
```

private variables

caller-supplied buffer

only called conditionally

Dual EC output masked by this

ScreenOS v.6.2

```
1. char seed[8] // dual_ec output
2. char block[8]; // X9.31 temporary output
3. char output[32]; // prng_generate final output
4. unsigned int index;
5.
6. void prng_reseed(void) {
7.     dual_ec_generate(output, 32);
8.     memcpy(seed, output, 32);
9.     index = 32;
10. }
11.
12. void prng_generate(void) {
13.     index = 0;
14.
15.     prng_reseed();
16.
17.     for (; index < 32; index += 8) {
18.         ...
19.         x9_31_generate_block(seed, block);
20.         ...
21.         memcpy(&output[index], block, 8);
22.     }
23. }
```

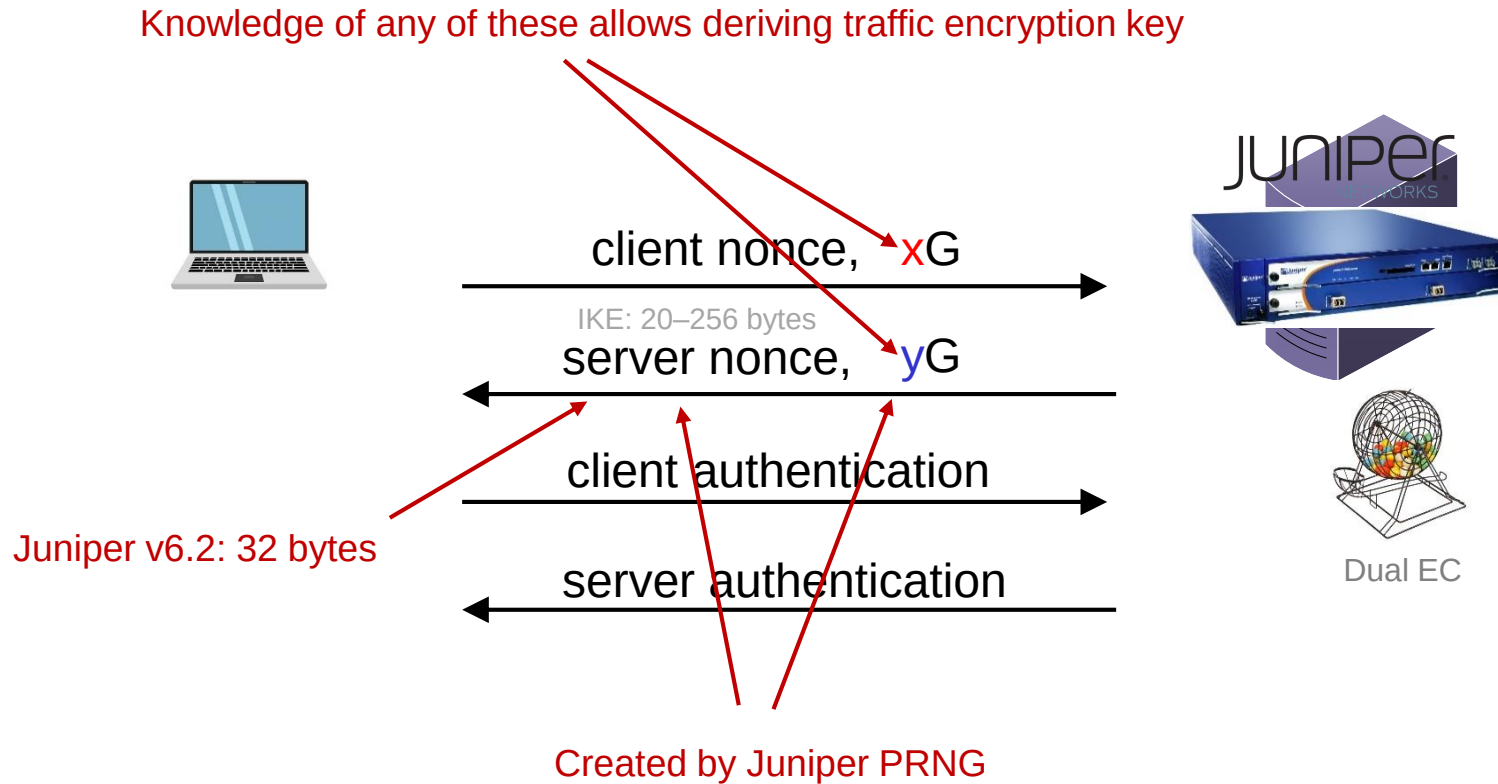
global variable

called unconditionally

global shared buffer

...means this is skipped

IKEv2 / IPsec



Juniper Networks backdoor

- Juniper Networks: big manufacturer of network equipment (routers, VPNs, firewalls, etc.)
 - Major customers: telcos, banks, US DoD

- **2015:**

IMPORTANT JUNIPER SECURITY ANNOUNCEMENT

During a recent internal code review, Juniper discovered **unauthorized code in ScreenOS** that could allow a knowledgeable attacker to gain administrative access to NetScreen® devices and to decrypt VPN connections.

- Hackers had obtained access to source code repository
- Only one change:

```
--- Qx = 2c55e5e45edf713dc43475effe8813a60326a64d9ba3d2e39cb639b0f3b0ad10  
+++ Qx = 9585320eeaf81044f20d55030a035b11bece81c785e6c933e4a8a131f6578107
```

Juniper created Q (yay! No NSA)

Who dis?

JUNIPER[®]
NETWORKS



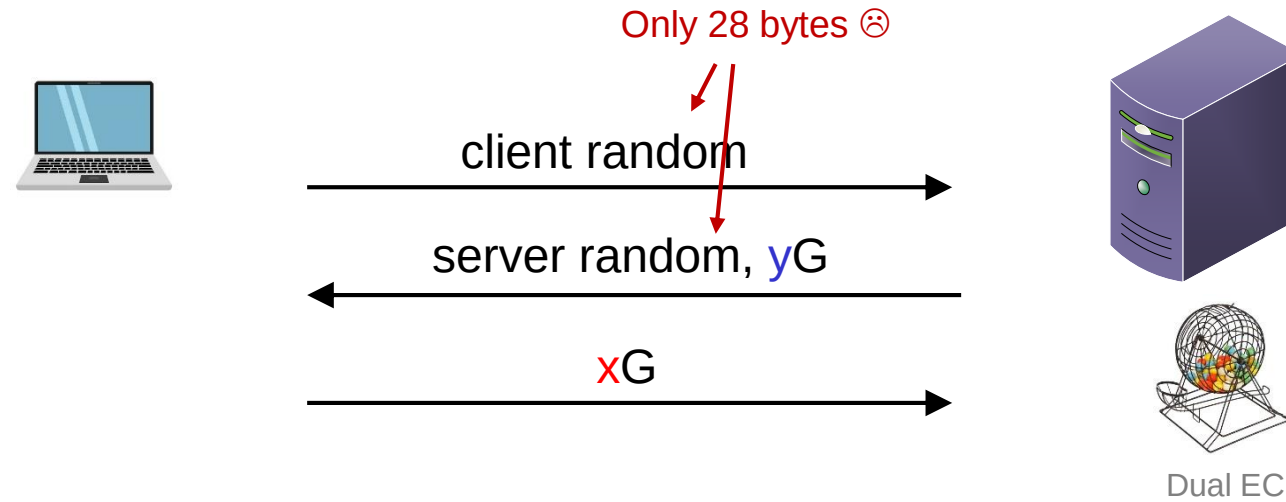
Juniper Networks backdoor

- **2008** – Juniper starts using Dual EC in ScreenOS
- **2012** – Someone hacks into Juniper's code repositories
 - Changes Q point in Dual EC
- **2015** – Juniper discovers intrusion
 - Changes Q back to its original value

JUNIPER[®]
NETWORKS



TLS 1.2



Extended Random

- NSA: please make the TLS nonces bigger...for reasons ;-)

- Implementing Extended Random makes exploiting Dual EC 10,000 times easier

- No real cryptographic justification exists for making them longer

[Search] [txt|pdf|bibtex] [Tracker] [Email] [Nits]

Versions: [00](#)
Network Working Group
Internet-Draft
Expires: June 16, 2007

E. Rescorla
Network Resonance
M. Salter
National Security Agency
December 13, 2006

Opaque PRF Inputs for TLS
draft-rescorla-tls-opaque-prf-input-00.txt

[Search] [txt|xml|pdf|bibtex] [Tracker] [Email] [Nits]

Versions: [00](#) [01](#) [02](#)
Network Working Group
Internet-Draft
Intended status: Informational
Expires: October 31, 2008

E. Rescorla
RTFM, Inc.
M. Salter
National Security Agency
April 29, 2008

Extended Random Values for TLS
draft-rescorla-tls-extended-random-00.txt

Status of this Memo

By submitting this Internet-Draft, each author represents that any applicable patent or other IPR claims of which he or she is aware will be disclosed, in accordance with [Section 6 of BCP 79](#).

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents.

[Search] [txt|pdf|bibtex] [Tracker] [Email] [Diff1] [Diff2] [Nits]

Versions: [00](#) [01](#)
Network Working Group
Internet-Draft
Intended status: Informational
Expires: April 27, 2010

J. Solinas
National Security Agency
P. Hoffman
VPN Consortium
October 24, 2009

Additional PRF Inputs for TLS
draft-solinas-tls-additional-prf-input-01

[Search] [txt|pdf|bibtex] [Tracker] [WG] [Email] [Diff1] [Diff2] [Nits]

Versions: [00](#) [01](#)
Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 15, 2010

Standards Track
P. Hoffman
VPN Consortium
February 11, 2010

Additional Random Extension to TLS
draft-hoffman-tls-additional-random-ext-01

Abstract

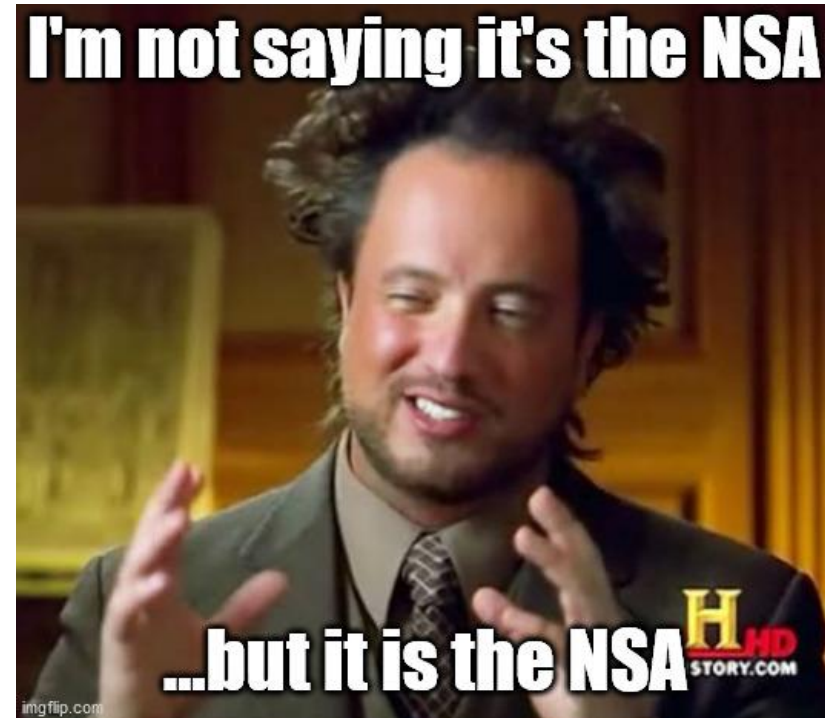
full conformance with the document may contain material published or made publicly available by a person(s) controlling the document. If you have granted the IETF license to such material outside the IETF, you must include an adequate license from such materials, this

RSA Security – BSAFE

- Turns out Juniper weren't the only one using Dual EC
- RSA Security
 - Big computer and network security company
 - Creator of BSAFE cryptographic library



- **2004** – accepted \$10 million from the NSA in order to make Dual EC the *default* in BSAFE
- **2014** – adapted the TLS Extended Random extension



Next week

- More Diffie-Hellman
- Implementation issues