Attacks against real-world cryptography

Nadia Heninger

UPenn → MSR New England → UC San Diego

March 20, 2019

Textbook RSA Encryption

[Rivest Shamir Adleman 1977]

Public Key

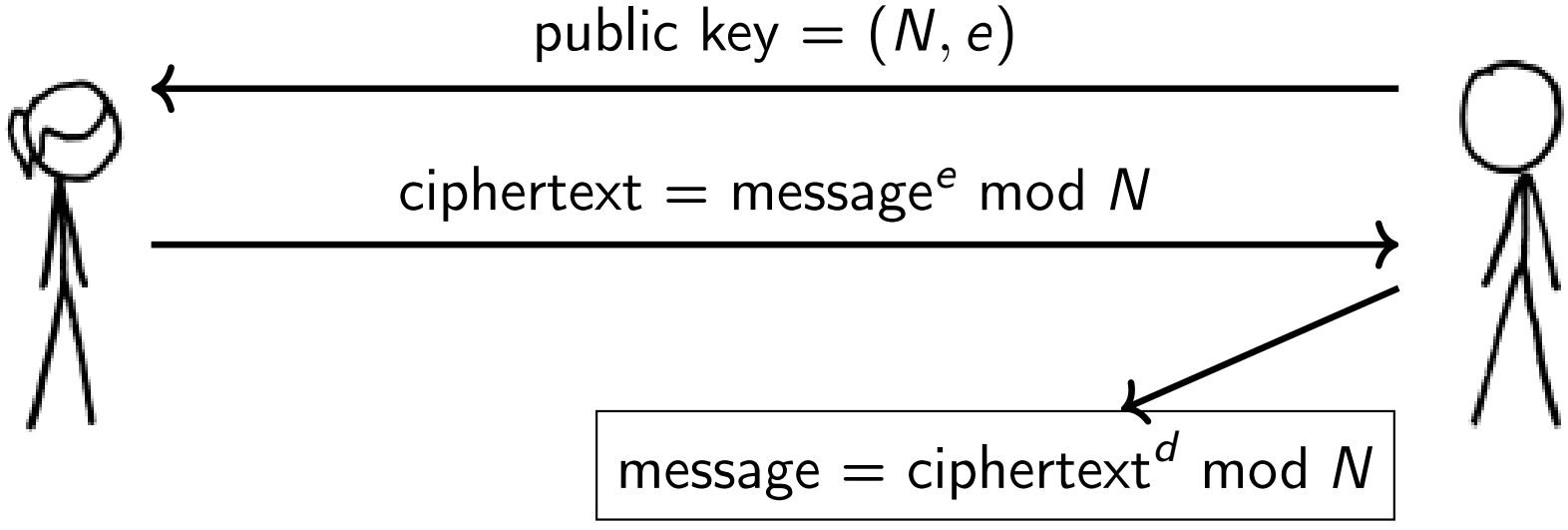
$N = pq$ modulus

e encryption exponent

Private Key

p, q primes

d decryption exponent
$(d = e^{-1} \bmod (p-1)(q-1))$



RSA cryptanalysis: computational problems

Factoring

Problem: Given N, compute its prime factors.

- Computationally equivalent to computing private key d.
- Factoring is in NP and coNP $\rightarrow$ not NP-complete (unless P=NP or similar).

eth roots mod N

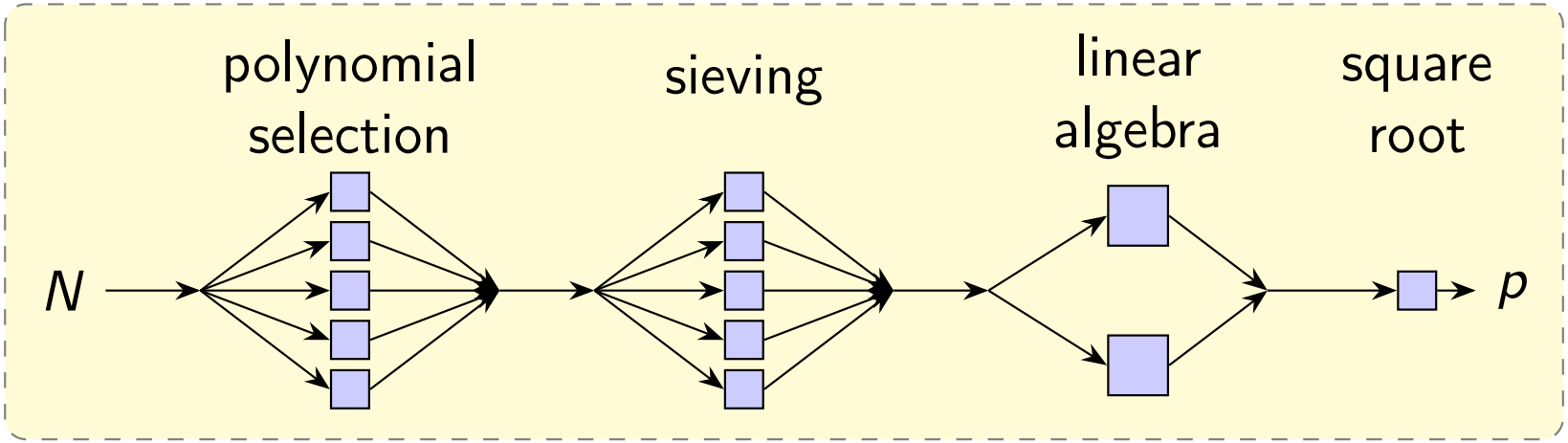
Problem: Given N, e, and c, compute x such that $x^e \equiv c \bmod N$.

- Equivalent to decrypting an RSA-encrypted ciphertext.
- Not known whether it is equivalent to factoring.

Factoring the hard way.

Factoring with the number field sieve

[Pollard], [Pomerance], [Lenstra,Lenstra]

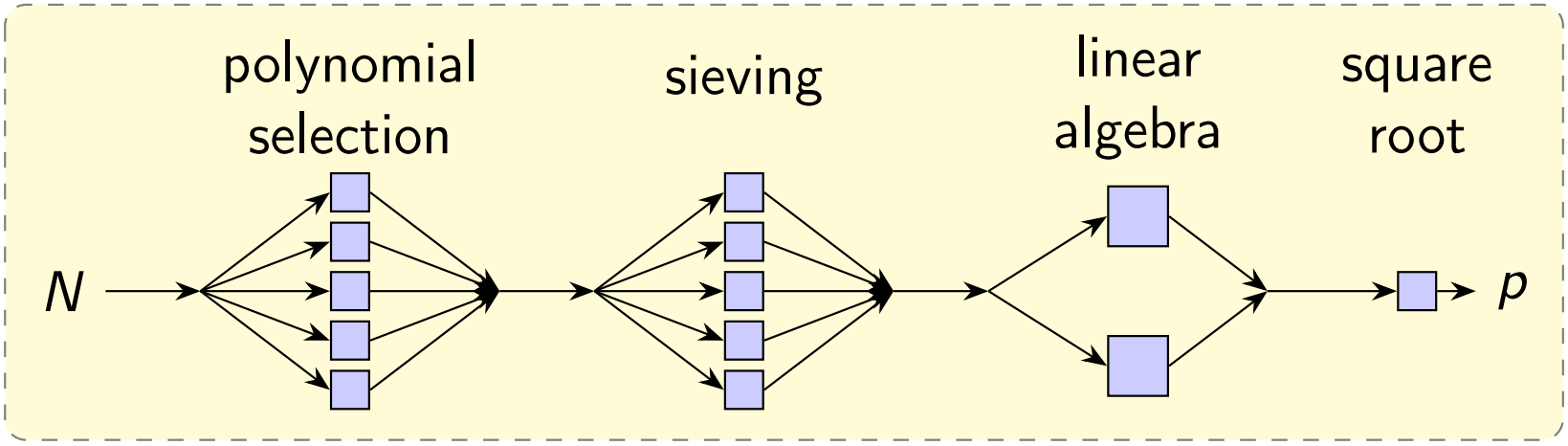


Algorithm

Motivation: Find a, b with $a^2 \equiv b^2 \bmod N$ and $\gcd(a+b, N)$ or $\gcd(a-b, N)$ nontrivial.

1. **Polynomial selection** Find a good choice of number field K.
2. **Relation finding** Factor elements over $\mathcal{O}_K$ and over $\mathbb{Z}$.
3. **Linear algebra** Find a square in $\mathcal{O}_K$ and a square in $\mathbb{Z}$.
4. **Square roots** Take square roots, map into $\mathbb{Z}$, and hope we find a factor.

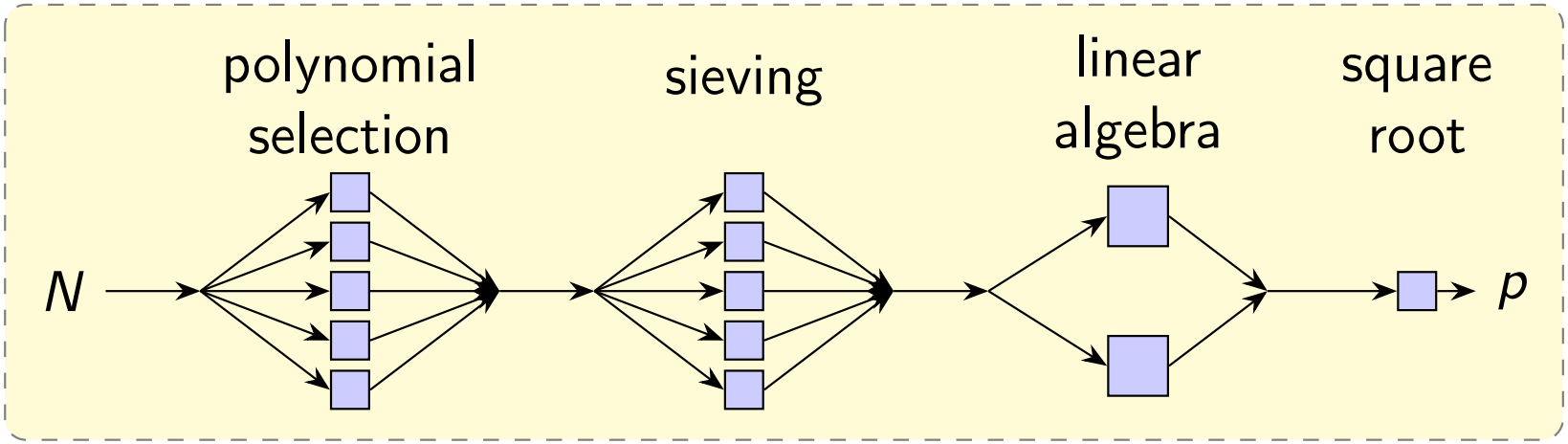
How long does it take to factor integers?



Answer 1: Asymptotic complexity.

$$L(1/3, 1.923) = \exp(1.923(\log N)^{1/3}(\log\log N)^{2/3})$$

How long does it take to factor integers?



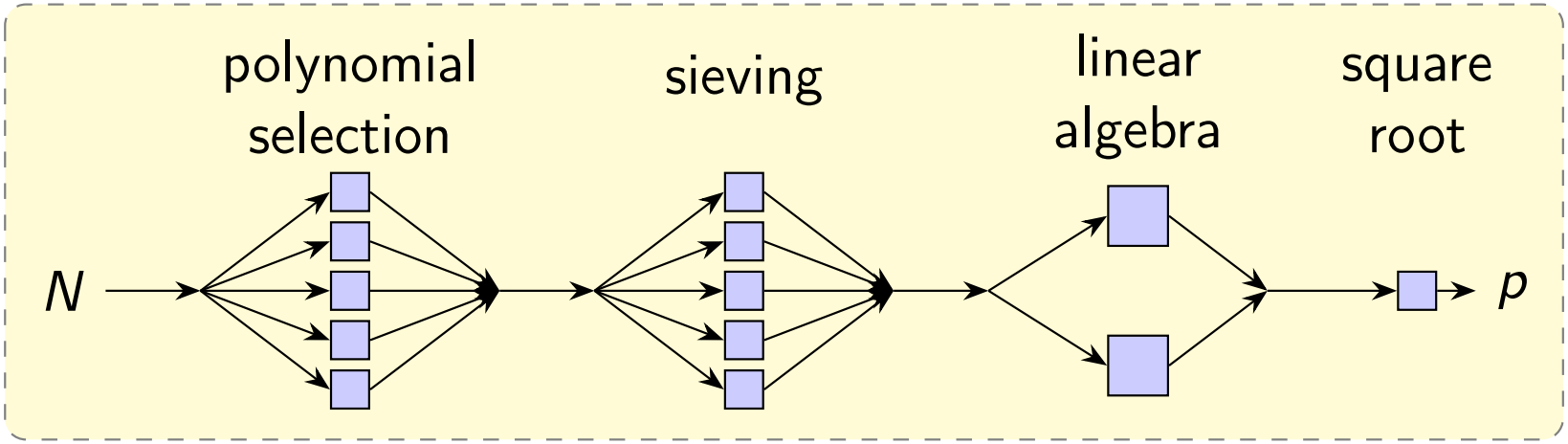
Answer 1: Asymptotic complexity.

$$L(1/3, 1.923) = \exp(1.923(\log N)^{1/3}(\log\log N)^{2/3})$$

Answer 2: Concrete records.

- In 1999, 512-bit RSA in 7 months and hundreds of computers. [Cavallar et al.]
- In 2009, 768-bit RSA in 2.5 calendar years. [Kleinjung et al.]

Answer 3: Extrapolation to larger key sizes.

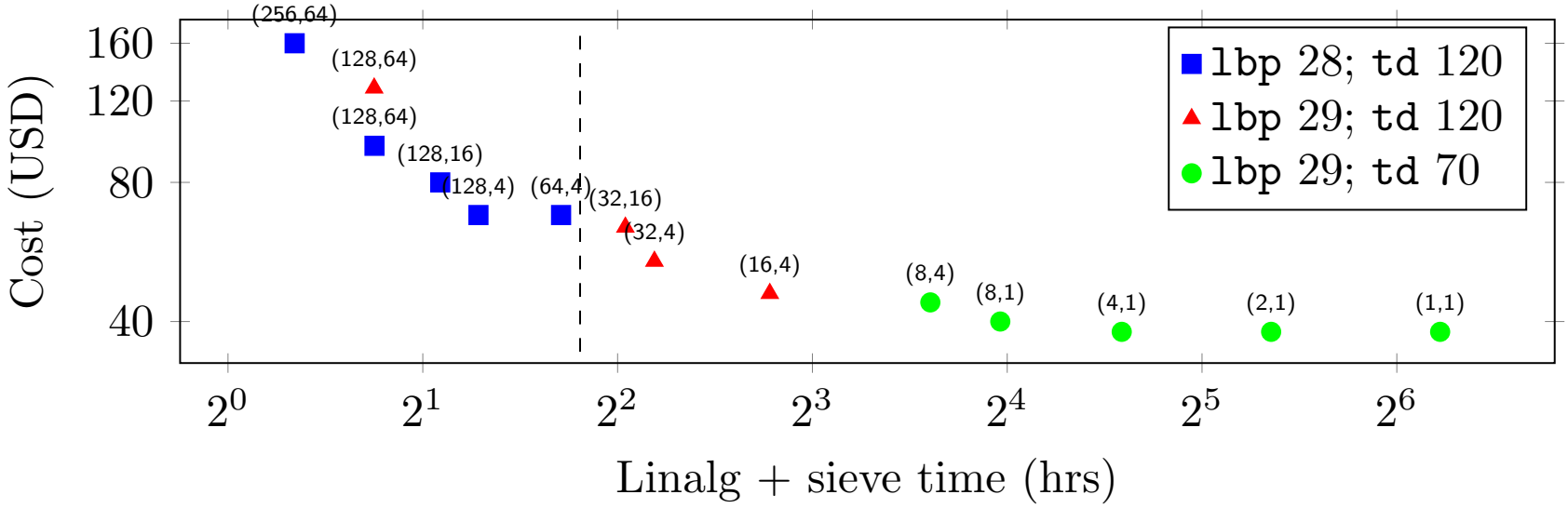


	Sieving			Linear Algebra	
	I	lpb	core-years	rows	core-years
RSA-512	14	29	0.5	4.3M	0.33
RSA-768	16	37	800	250M	100
RSA-1024	18	42	1,000,000	8.7B	120,000

Answer 4: Do it yourself.

[VCLFBH 2016]

Time to factor 512-bit RSA on Amazon EC2 in 2016:



Practical uses for 512-bit factorization

- 2009: Benjamin Moody factors 512-bit TI calculator code signing key.
- 2012: Zach Harris factors Google's 512-bit DKIM key.
- 2015: **FREAK attack** [BBDFKPSZ 2015]
  Modern TLS connections can be downgraded to 512-bit export-grade RSA. 10% of popular HTTPS sites vulnerable.
- 2016: Many 512-bit DNSsec and DKIM keys still in the wild.
- 2016–2018: TeslaCrypt, Chainshot Ransomware use 512-bit RSA keys.

Factoring the easy way.

RSA and GCDs

If two RSA moduli share a common factor,

$$N_1 = pq_1 \qquad\qquad N_2 = pq_2$$

RSA and GCDs

If two RSA moduli share a common factor,

$$N_1 = pq_1 \qquad N_2 = pq_2$$

$$\gcd(N_1, N_2) = p$$

You can factor both keys with GCD algorithm.

Time to factor
768-bit RSA modulus:
2.5 calendar years
[Kleinjung et al. 2010]

Time to calculate GCD
for 1024-bit RSA moduli:
15μs

Should we expect to find prime collisions in the wild?

Experiment: Compute GCD of each pair of M RSA moduli randomly chosen from P primes.

What *should* happen? **Nothing.**

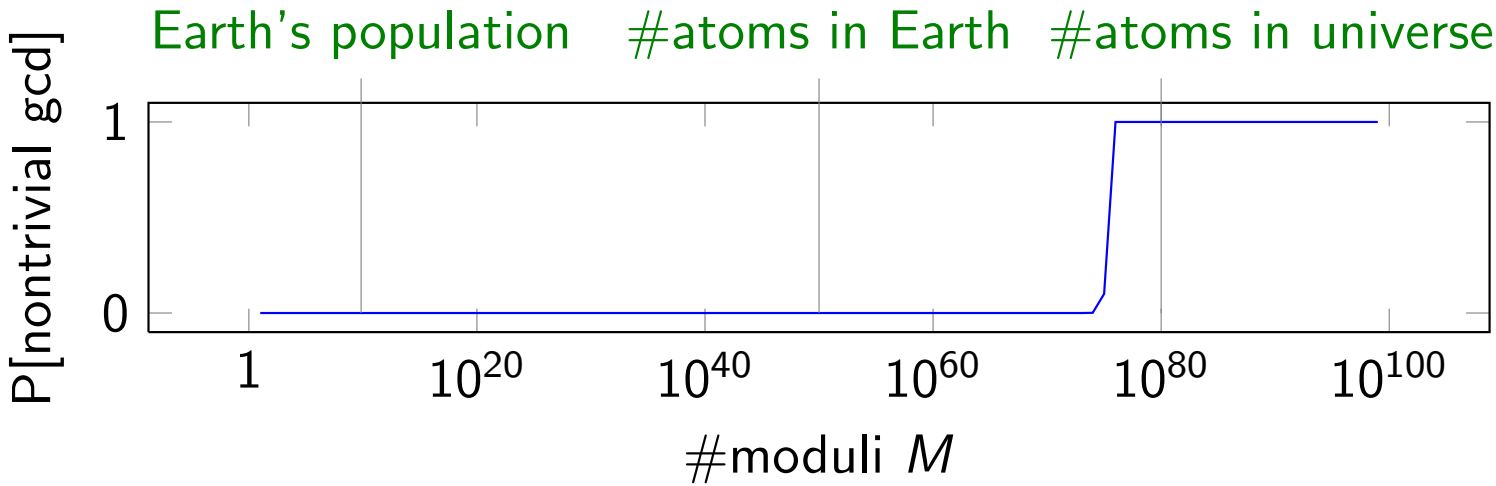
Should we expect to find prime collisions in the wild?

Experiment: Compute GCD of each pair of M RSA moduli randomly chosen from P primes.

What *should* happen? **Nothing.**

Prime Number Theorem:
$\sim 10^{150}$ 512-bit primes

Birthday bound:
$\Pr[\text{nontrivial gcd}] \approx 1 - e^{-2M^2/P}$



What happened when we GCDed RSA keys in 2012?

Computed private keys for

- 64,081 HTTPS servers (0.50%).
- 2,459 SSH servers (0.03%).
- 2 PGP users (and a few hundred invalid keys).
  [Lenstra et al. 2012]

What happened when we GCDed RSA keys in 2012?

Computed private keys for

- 64,081 HTTPS servers (0.50%).
- 2,459 SSH servers (0.03%).
- 2 PGP users (and a few hundred invalid keys).
  [Lenstra et al. 2012]

What has happened since?

- 103 Taiwanese citizen smart card keys [Bernstein, Chang, Cheng, Chou, Heninger, Lange, van Someren 2013]
- 90 export-grade HTTPS keys.
  [Albrecht, Papini, Paterson, Villanueva-Polanco 2015]
- 313,330 HTTPS, SSH, IMAPS, POP3S, SMTPS keys
  [Hastings Fried Heninger 2016]
- 3,337 Tor relay RSA keys.
  [Kadianakis, Roberts, Roberts, Winter 2017]

Algorithmic note: How to efficiently compute pairwise GCDs

Computing pairwise $\gcd(N_i, N_j)$ the naive way on all of the unique RSA keys in a single set of Internet-wide scans would take

$$15\mu\text{s} \times \binom{14 \times 10^6}{2} \text{pairs} \approx 1100 \text{ years}$$

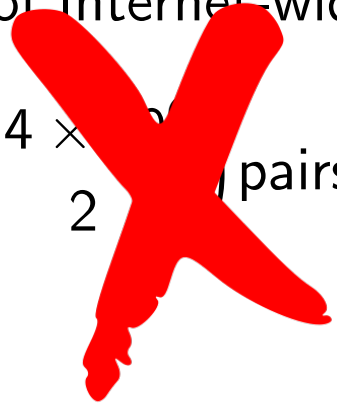
of computation time.

Algorithmic note: How to efficiently compute pairwise GCDs

Computing pairwise $\gcd(N_i, N_j)$ the naive way on all of the unique RSA keys in a single set of Internet-wide scans would take

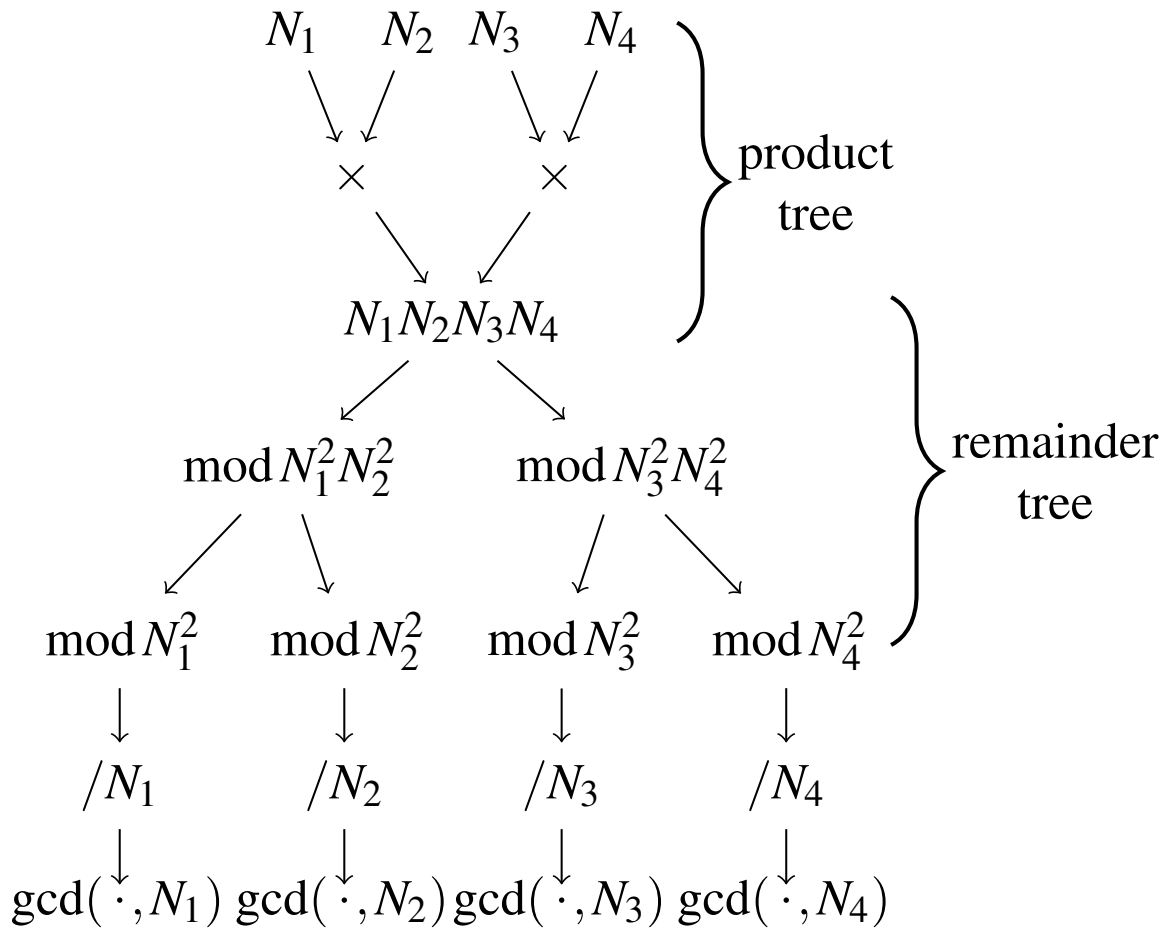
$$15\mu\text{s} \times \binom{14 \times 10^6}{2} \text{pairs} \approx 1100 \text{ years}$$

of computation time.



Algorithmic note: How to efficiently compute pairwise GCDs

Algorithm from (Bernstein 2004)



A few hours for 80M keys.

Implementation available at https://factorable.net.

Widespread RNG failures on low resource devices

We accidentally found *multiple independent implementation problems*.

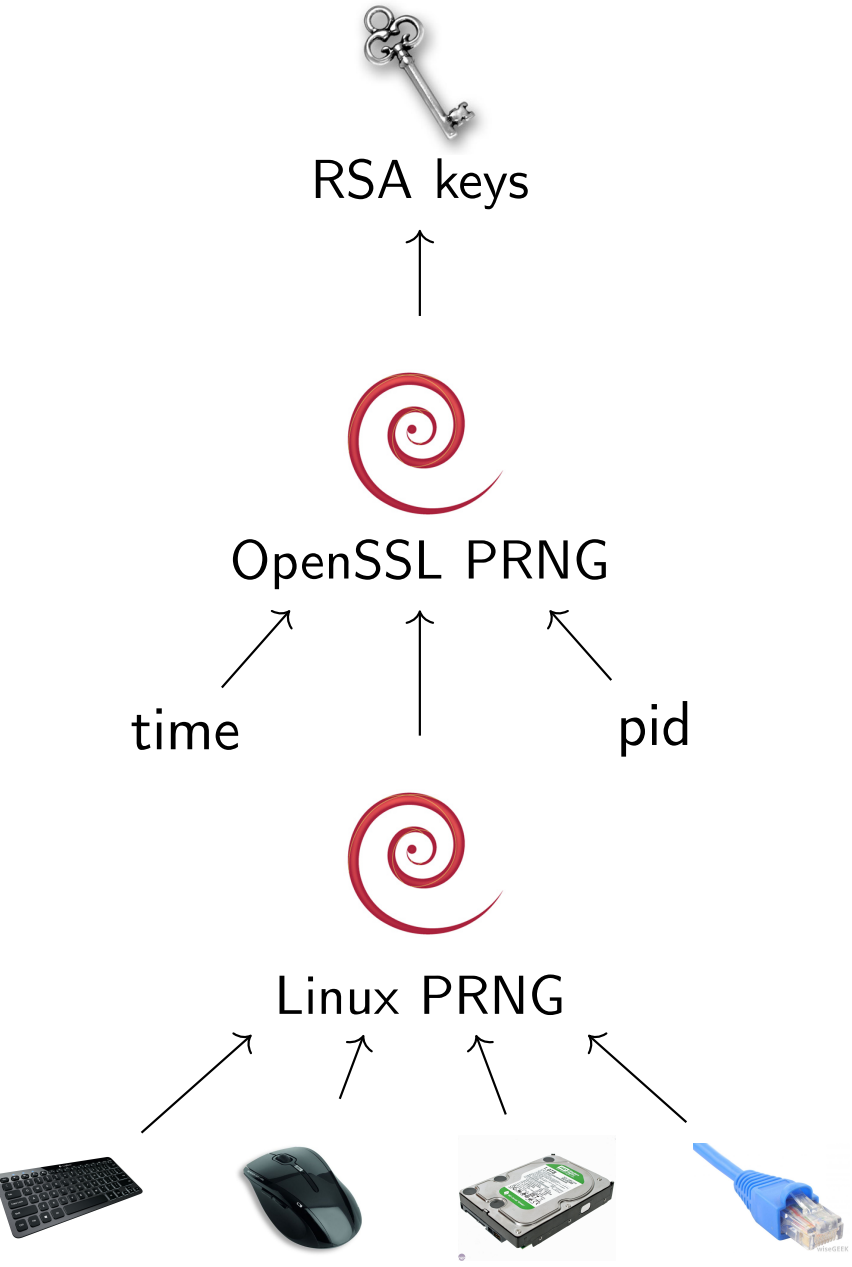
Clue #1: Vast majority of weak keys generated by low resource devices.

- Juniper network security devices
- Cisco routers
- Fortigate firewalls
- Intel server management cards
- ...

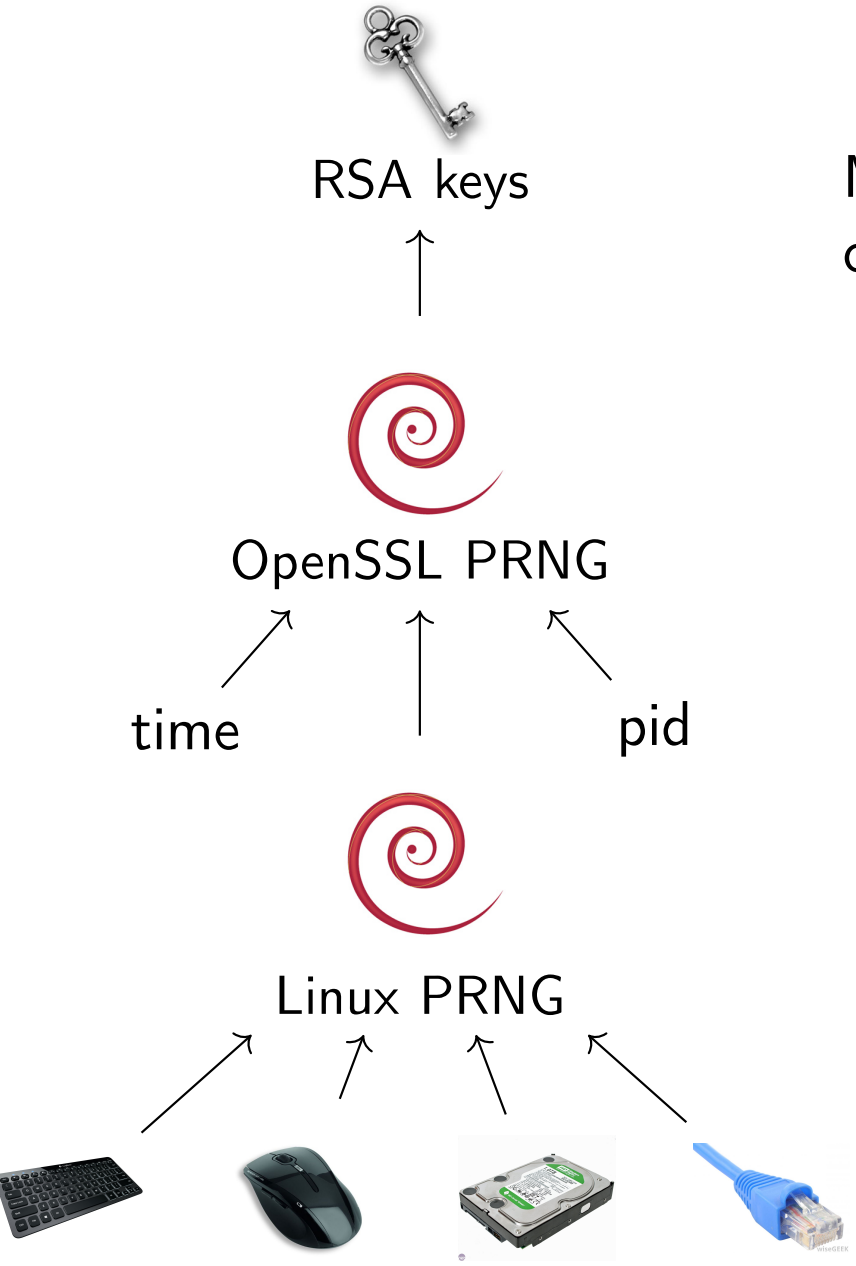
Identified devices from $>$ 50 manufacturers

Clue #2: Very different behavior for different devices. Different companies, implementations, underlying software, distributions of prime factors.

One cause: Cascading PRNG failures

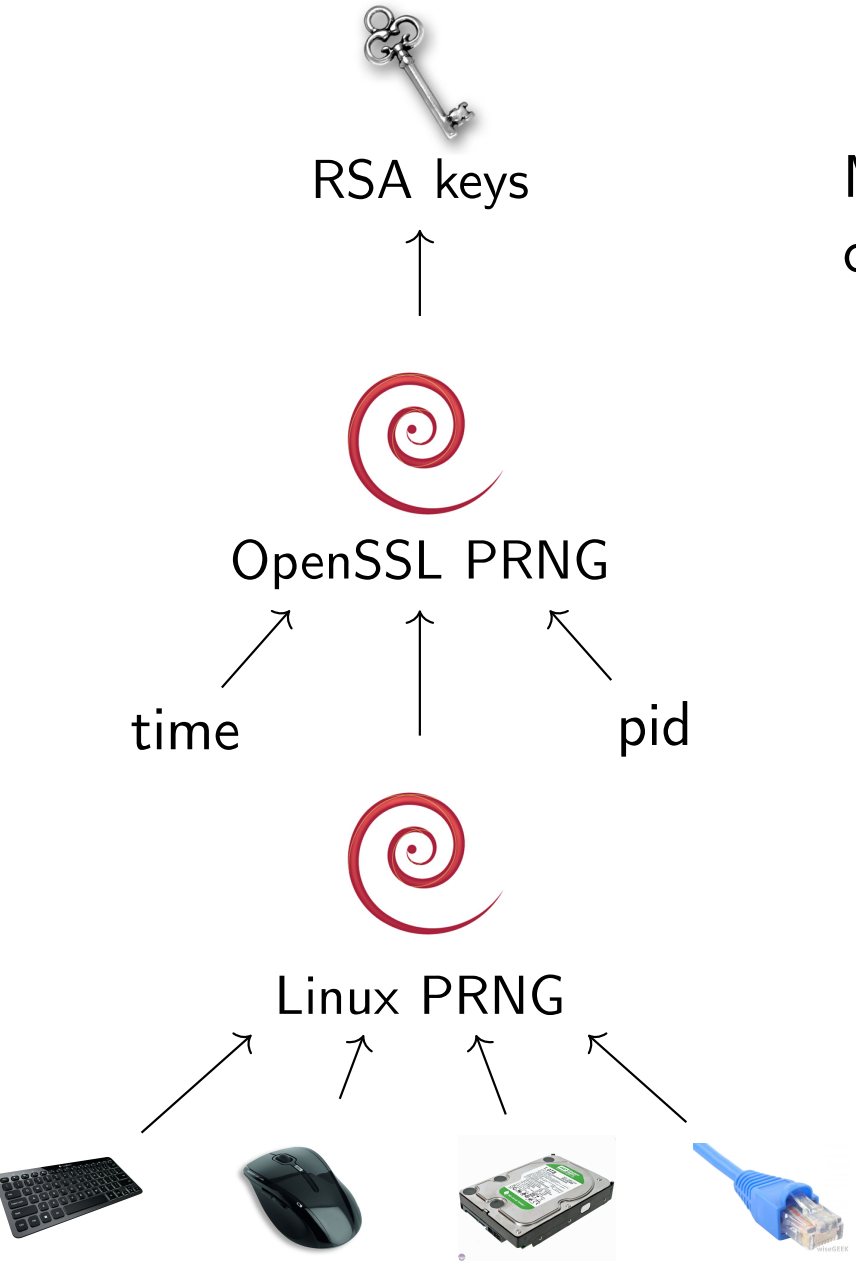


One cause: Cascading PRNG failures



Many devices automatically generate crypto keys on first boot.

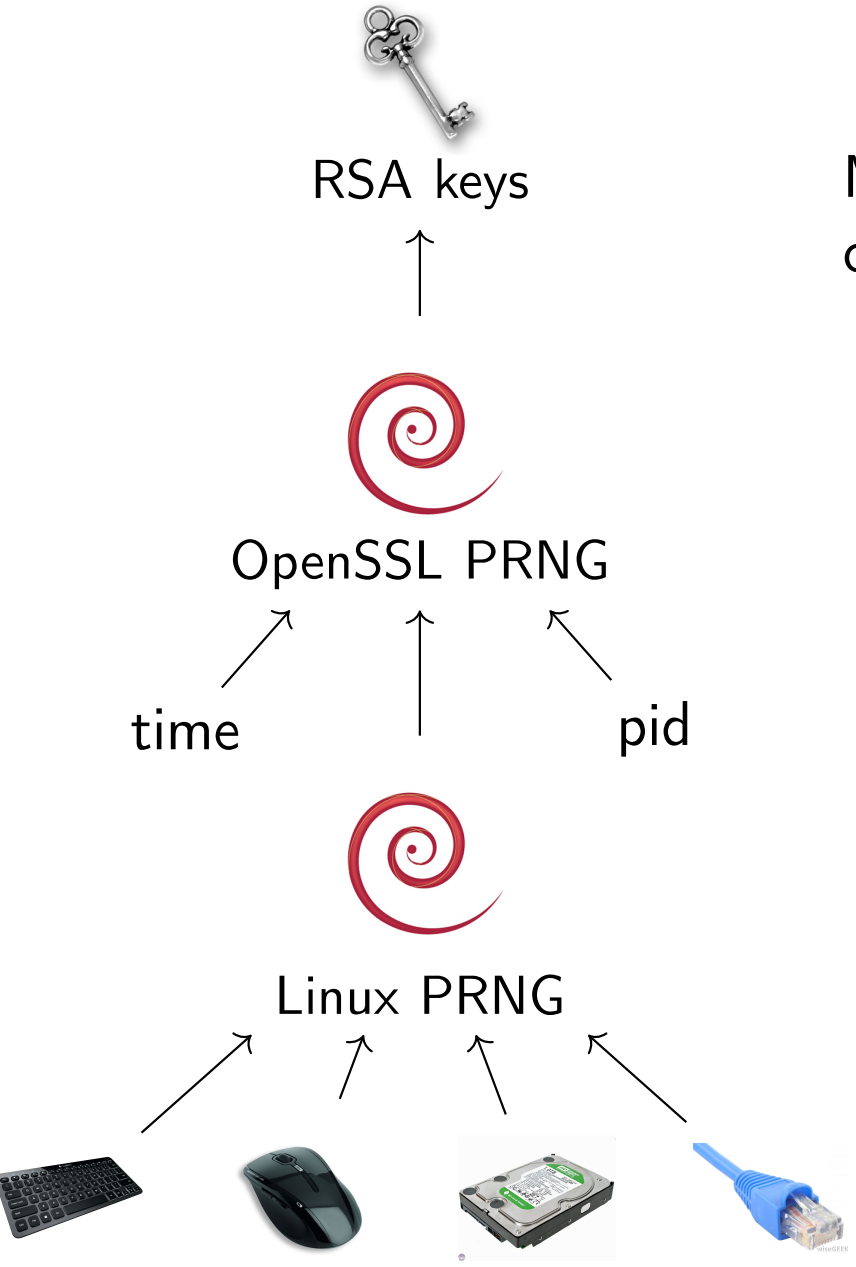
One cause: Cascading PRNG failures



Many devices automatically generate crypto keys on first boot.

- Headless or embedded devices often lack these entropy sources.

One cause: Cascading PRNG failures



Many devices automatically generate crypto keys on first boot.

- The Linux PRNG had not yet been seeded when queried by OpenSSL
  $\Longrightarrow$ deterministic output.
  Patched since 2012.

- Headless or embedded devices often lack these entropy sources.

How did factorable keys arise in practice?

- Usability problems in random number generator interface.

```
 /* We'll use /dev/urandom by default, since /dev/random is
too much hassle.  If system developers aren't keeping seeds
between boots nor getting any entropy from somewhere it's
their own fault.  */
#define DROPBEAR_RANDOM_DEV "/dev/urandom"
```

- A cascade of vulnerable software behaviors.
  - OpenSSL mixed current time in seconds into RNG state
  - This led to factorable and not merely repeated keys.

Generating vulnerable RSA keys in software

- Insufficiently random seeds for pseudorandom number generator $\implies$ we should see repeated keys.

```
prng.seed()
p = prng.random_prime()
q = prng.random_prime()
N = p*q
```

- We do:
  - $>$ 60% of hosts share keys
  - At least 0.3% due to bad randomness.
- Repeated keys may be a sign that implementation is vulnerable to a targeted attack.

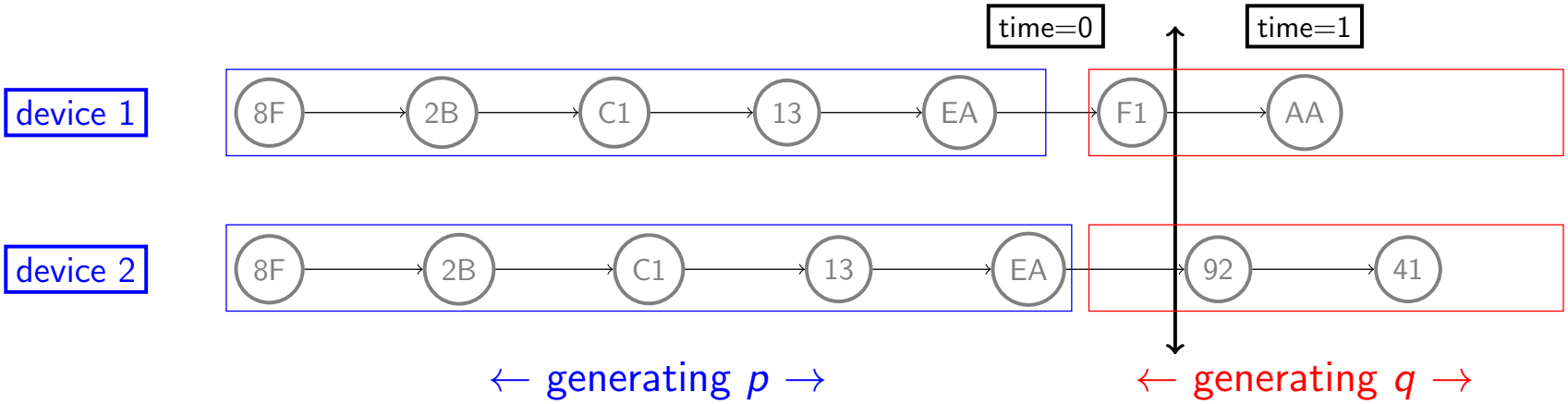
But why do we see factorable keys?

Generating factorable RSA keys in software

```
prng.seed()
p = prng.random_prime()
prng.add_randomness()
q = prng.random_prime()
N = p*q
```

OpenSSL adds time in seconds

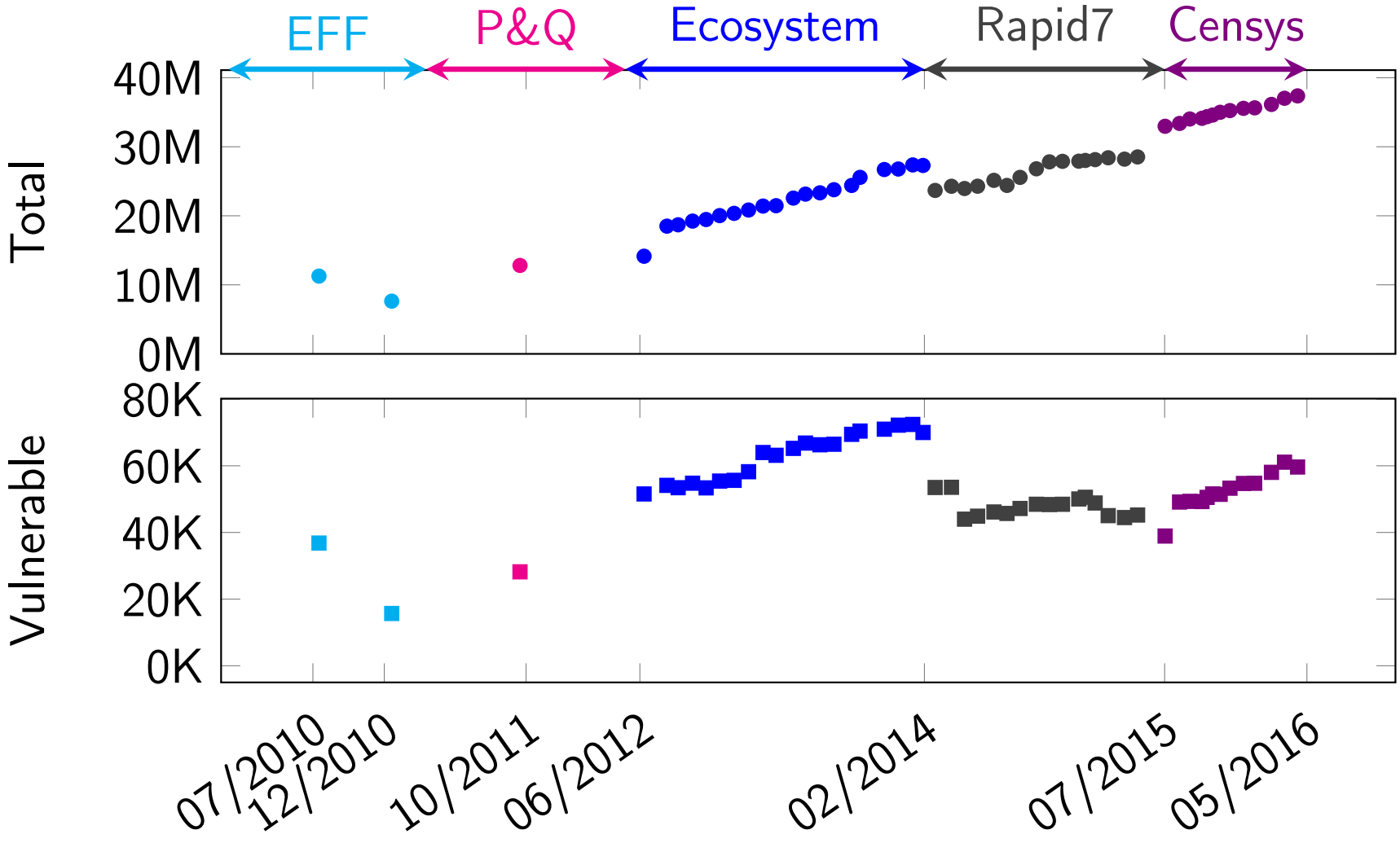
Insufficient randomness can lead to factorable keys.



Experimentally verified OpenSSL generates factorable keys in this situation.

Follow-up study: Six years of factoring keys

- 51 million distinct HTTPS RSA moduli : 0.43% vulnerable
- 65 million distinct HTTPS certificates : 2.2% vulnerable
- 1.5 billion HTTPS host records : 0.19% vulnerable

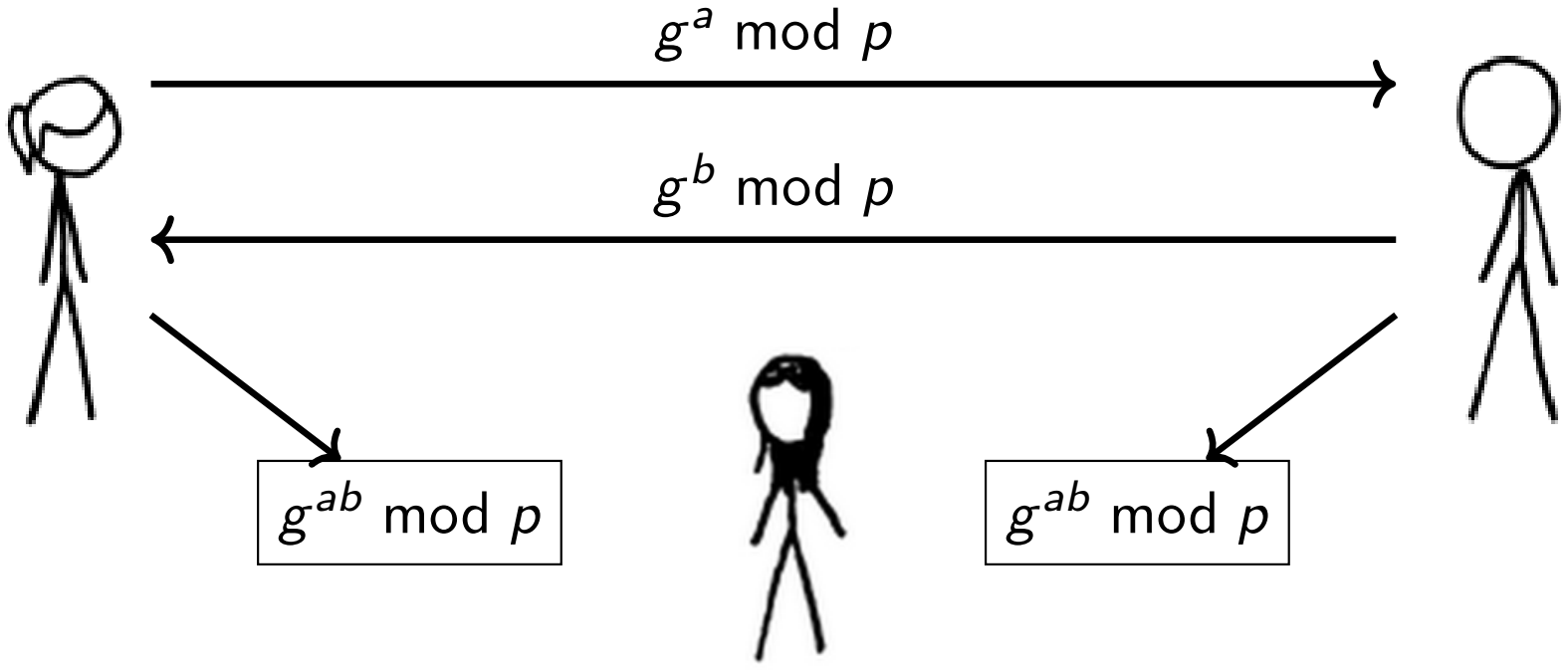


Textbook (Finite-Field) Diffie-Hellman Key Exchange

[Diffie Hellman 1976]

p a prime (so $\mathbb{F}_p^*$ is a cyclic group)

$g < p$ multiplicative group generator (often 2 or 5)



Diffie-Hellman cryptanalysis and computational problems

Discrete Log

Problem: Given g^a, compute a.

- Solving this problem permits attacker to compute shared key by computing a and raising $(g^b)^a$.
- Discrete log is in NP and coNP $\rightarrow$ not NP-complete (unless P=NP or similar).

Diffie-Hellman problem

Problem: Given g^a, g^b, compute g^{ab}.

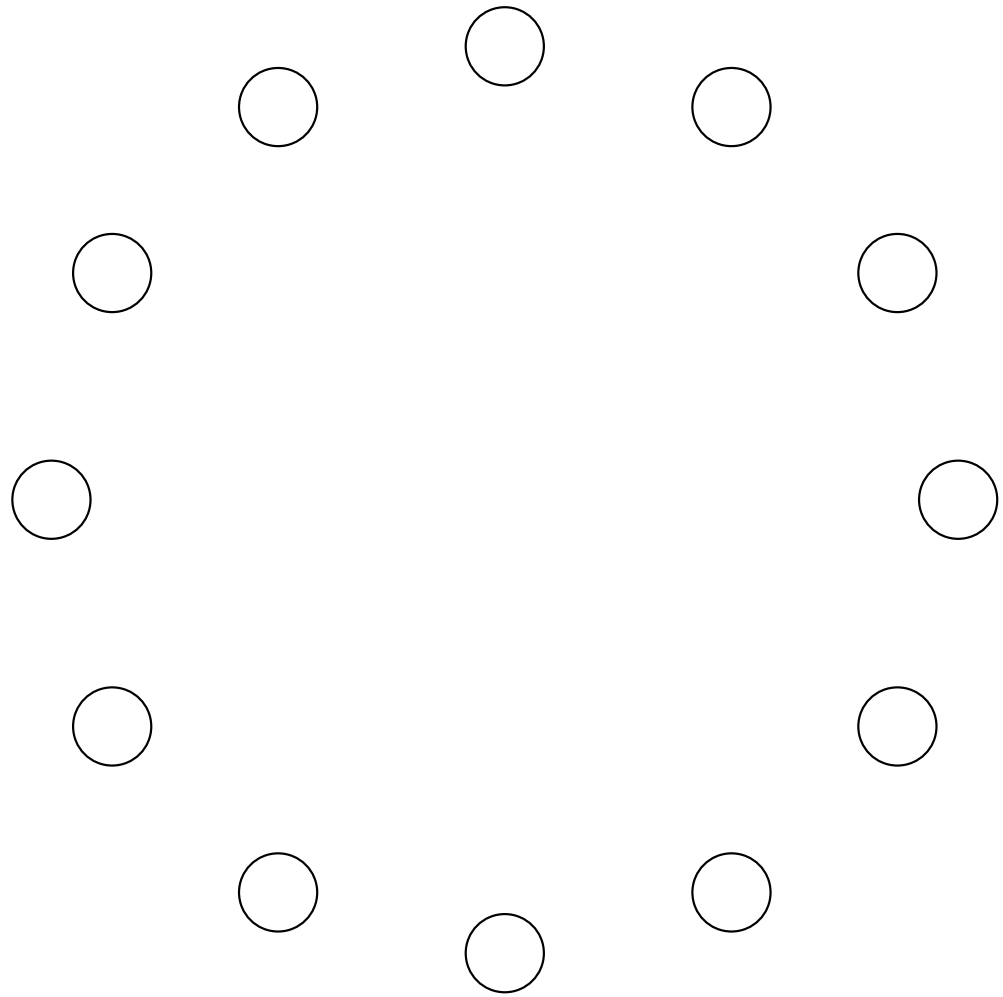
- Exactly problem of computing shared key from public information.
- Reduces to discrete log in some cases:
- (Computational) Diffie-Hellman assumption: This problem is hard in general.

Computing discrete logs the easy way.

Reminder: groups, subgroups, and generators

Cyclic group

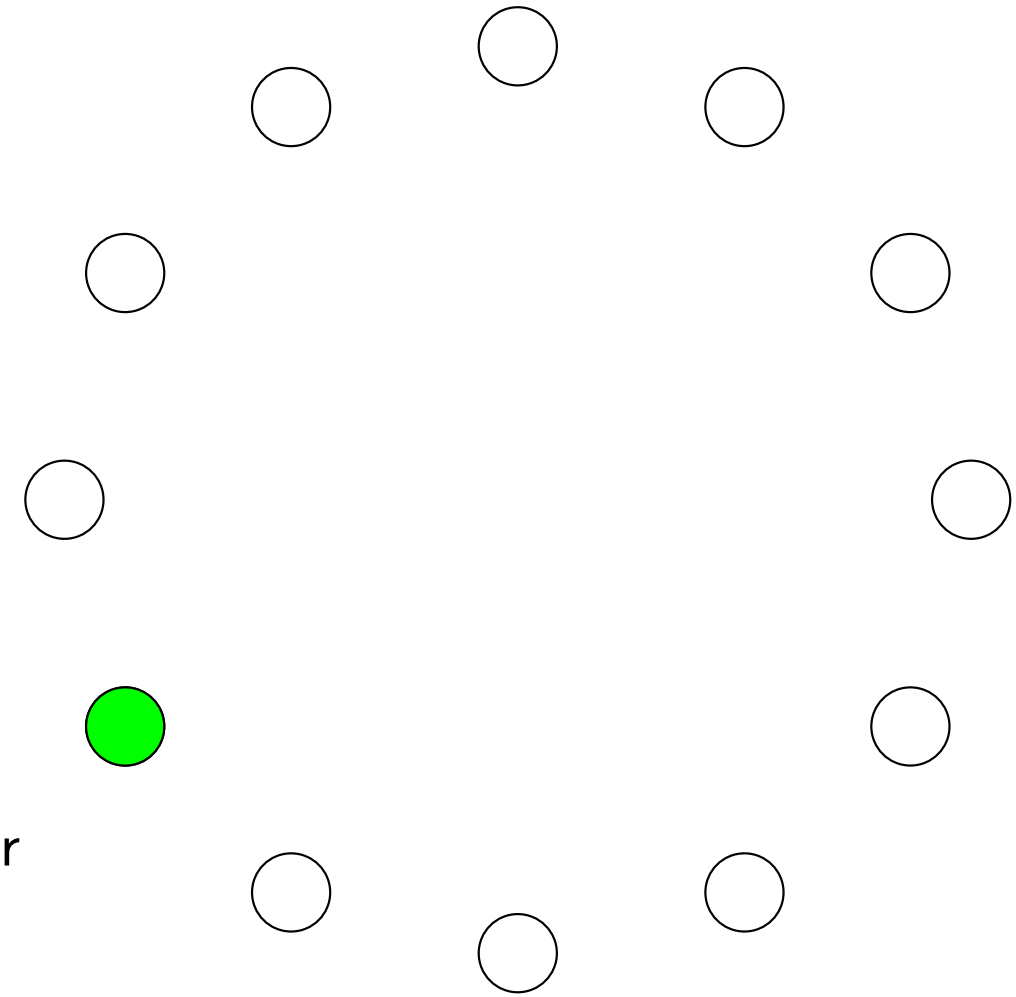
Order = #elements in group



Reminder: groups, subgroups, and generators

Cyclic group

Order = #elements in group

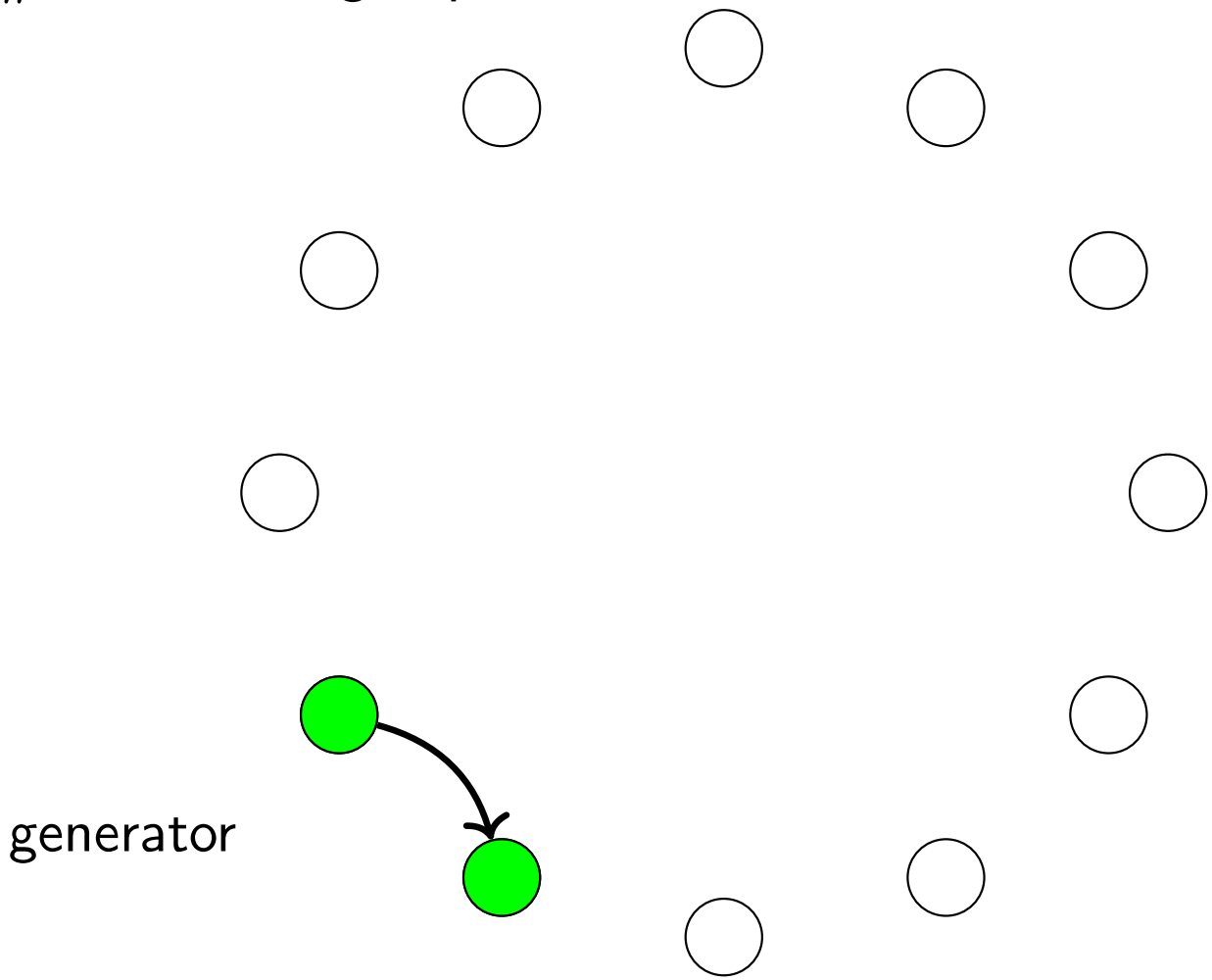


generator

Reminder: groups, subgroups, and generators

Cyclic group

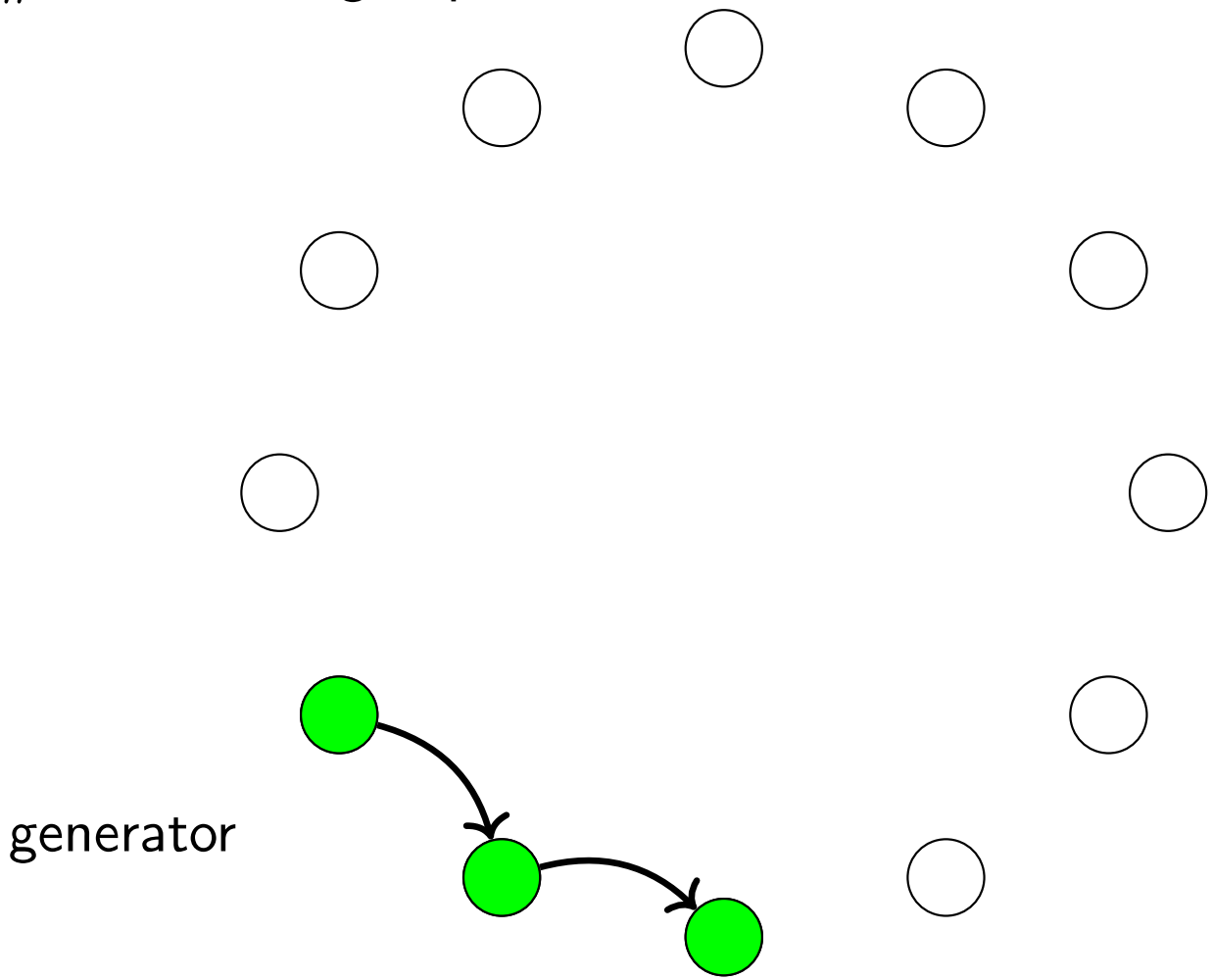
Order = #elements in group



Reminder: groups, subgroups, and generators

Cyclic group

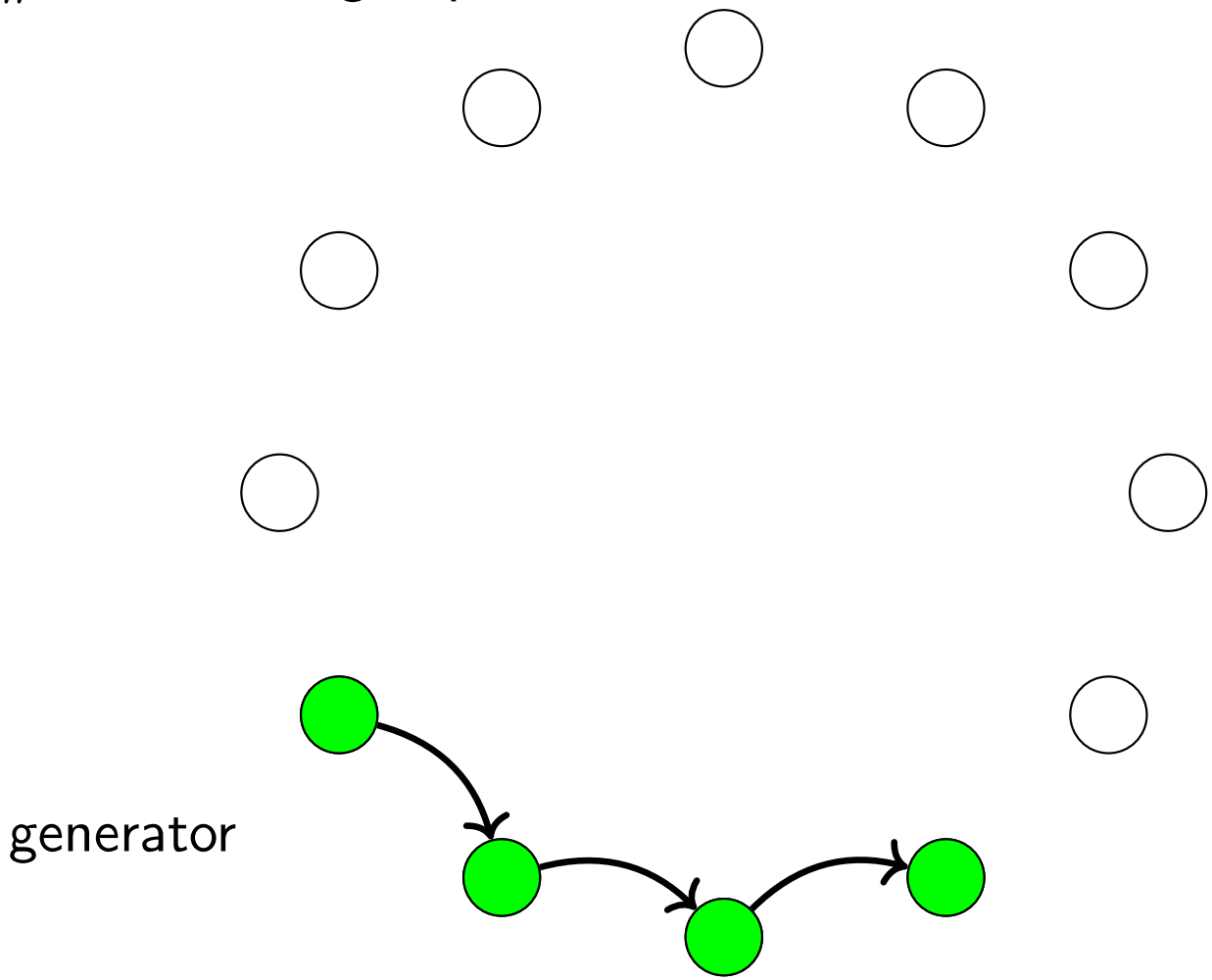
Order = #elements in group



Reminder: groups, subgroups, and generators

Cyclic group

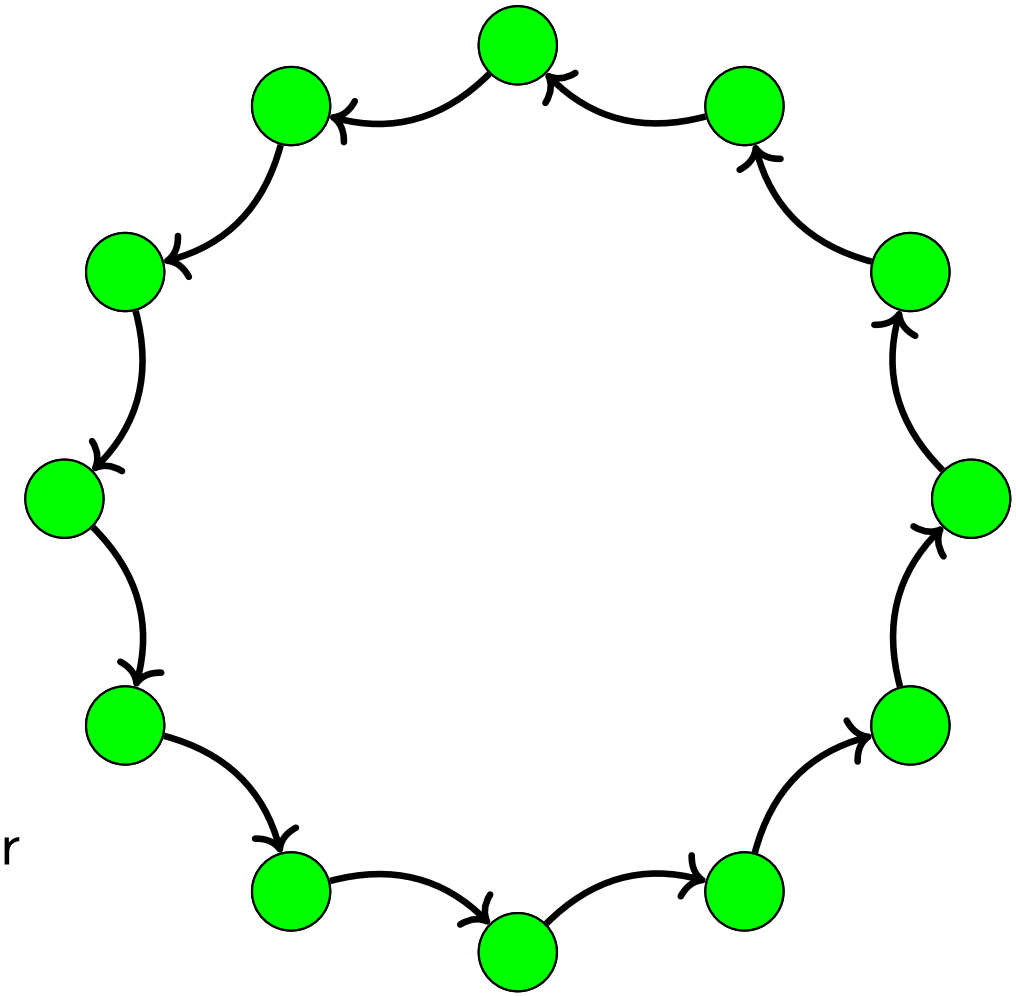
Order = #elements in group



Reminder: groups, subgroups, and generators

Cyclic group

Order = #elements in group



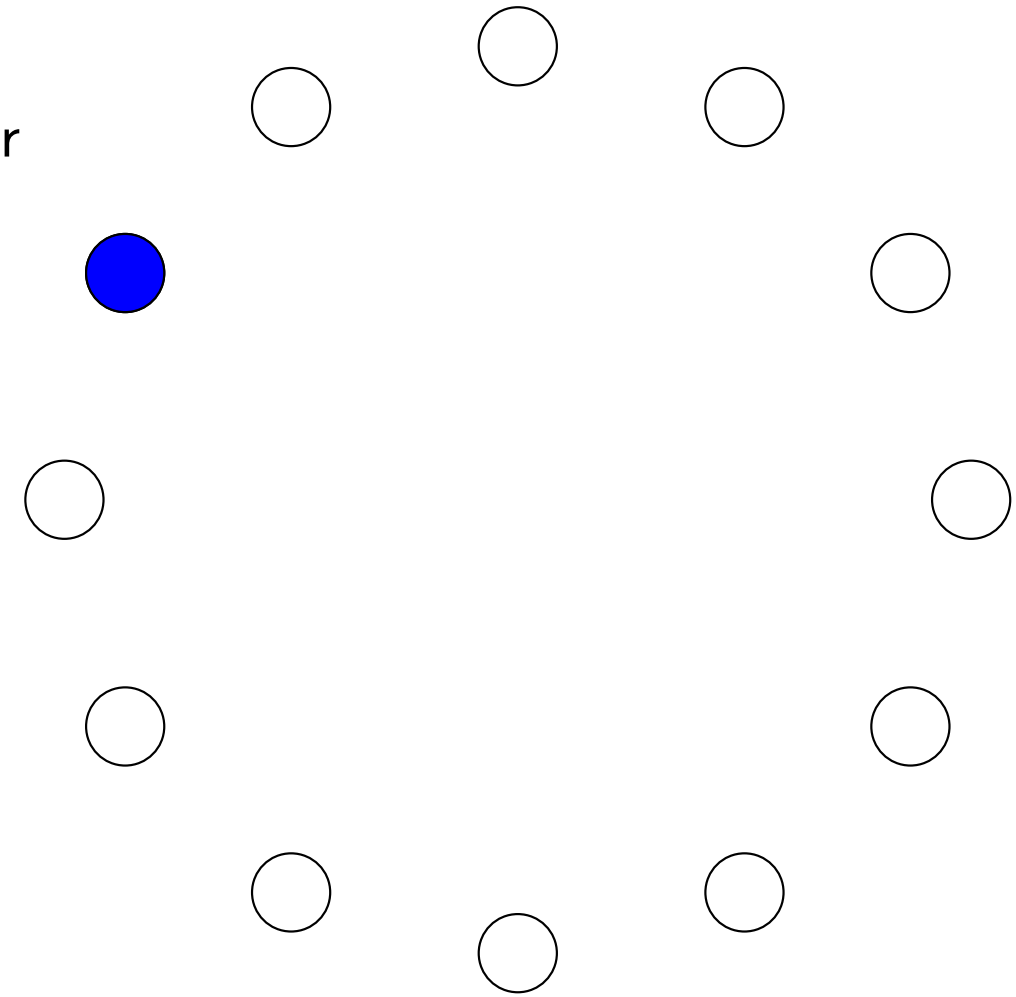
generator

Reminder: groups, subgroups, and generators

Subgroup

Order = #elements in subgroup

generator

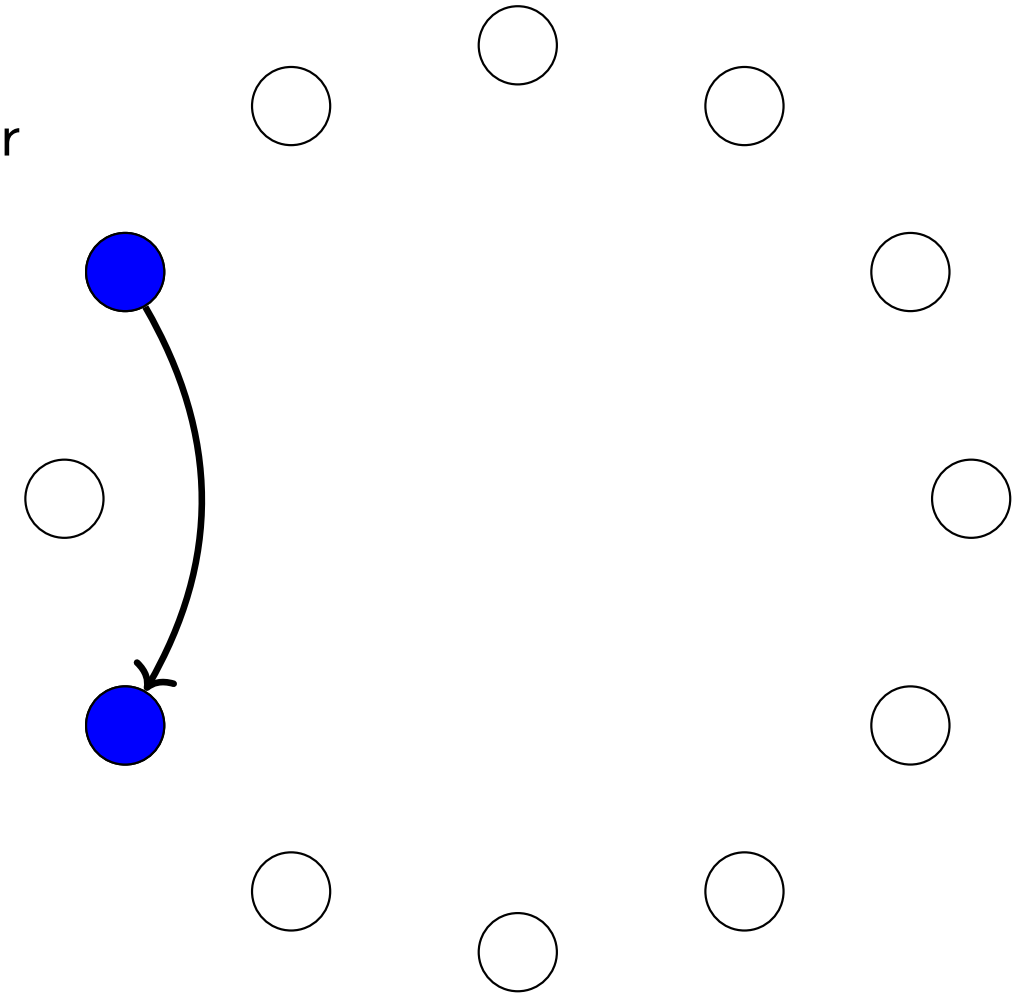


Reminder: groups, subgroups, and generators

Subgroup

Order = #elements in subgroup

generator

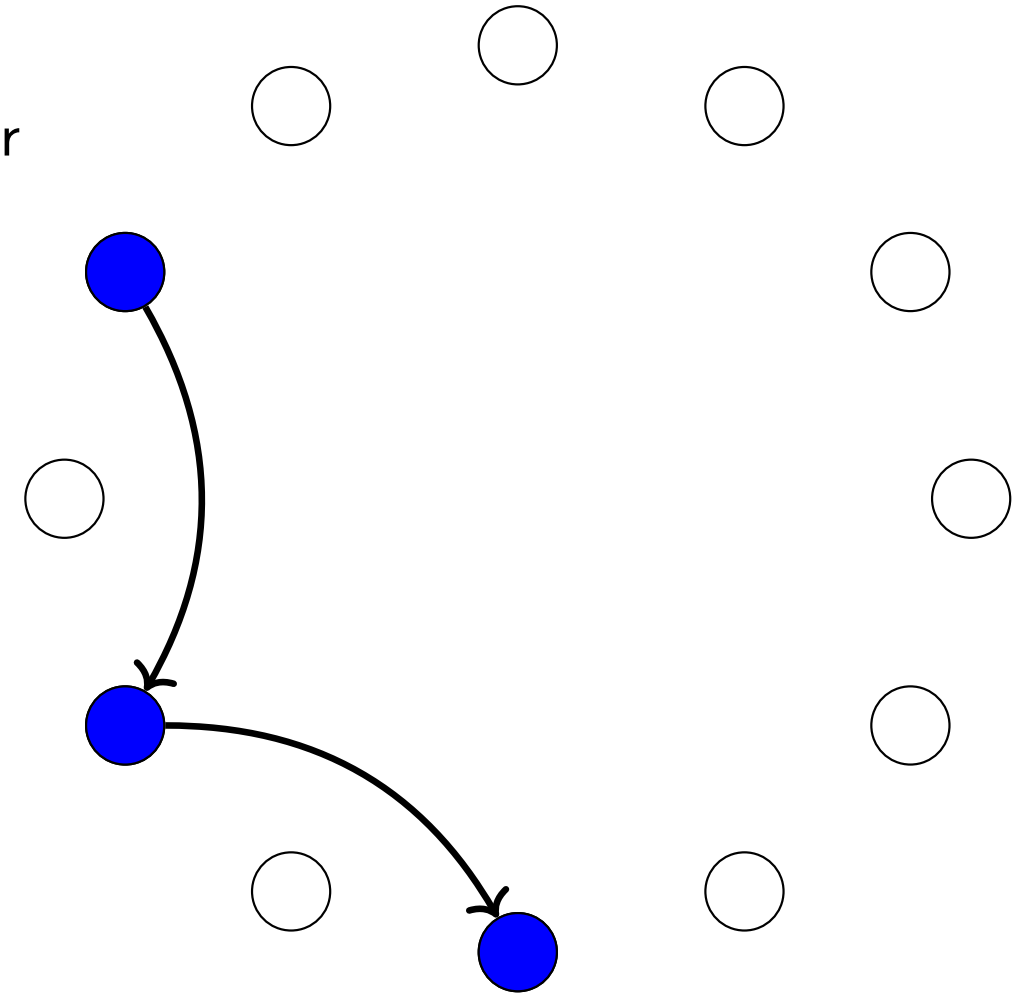


Reminder: groups, subgroups, and generators

Subgroup

Order = #elements in subgroup

generator

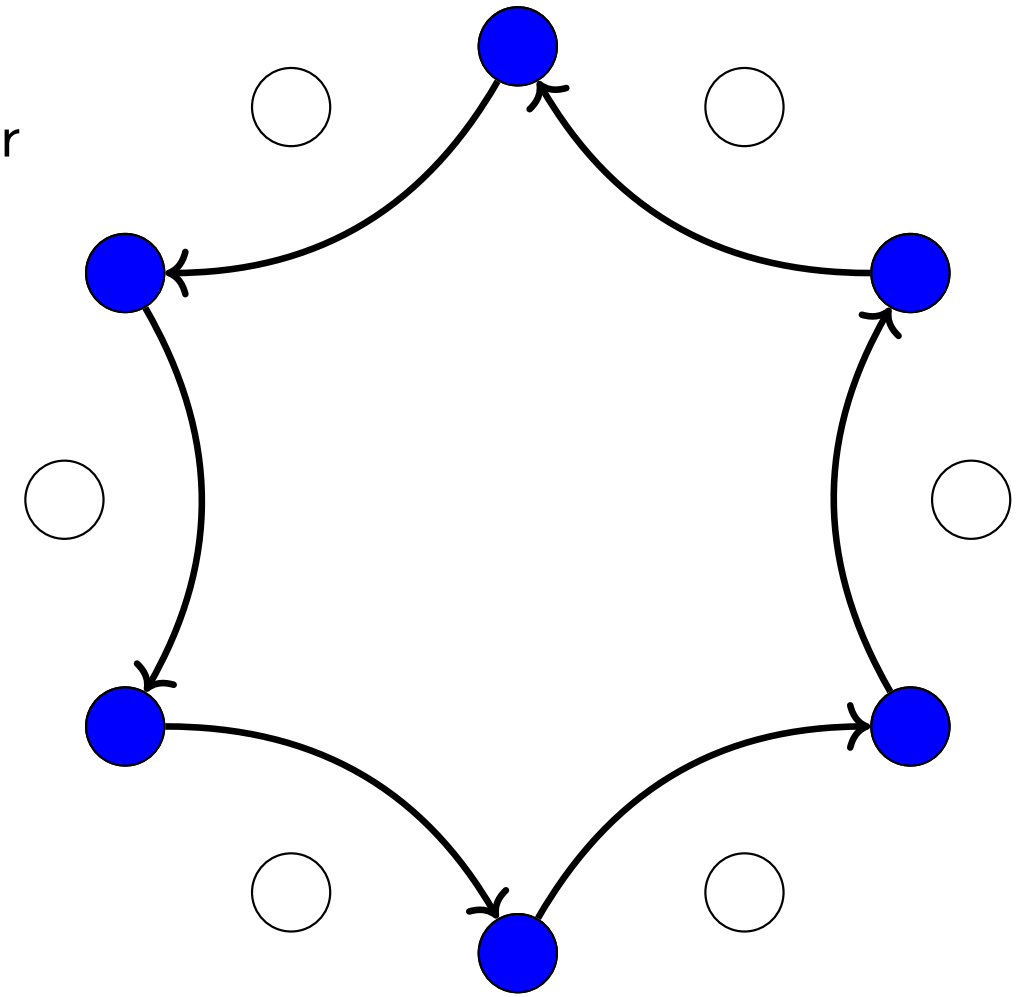


Reminder: groups, subgroups, and generators

Subgroup

Order = #elements in subgroup

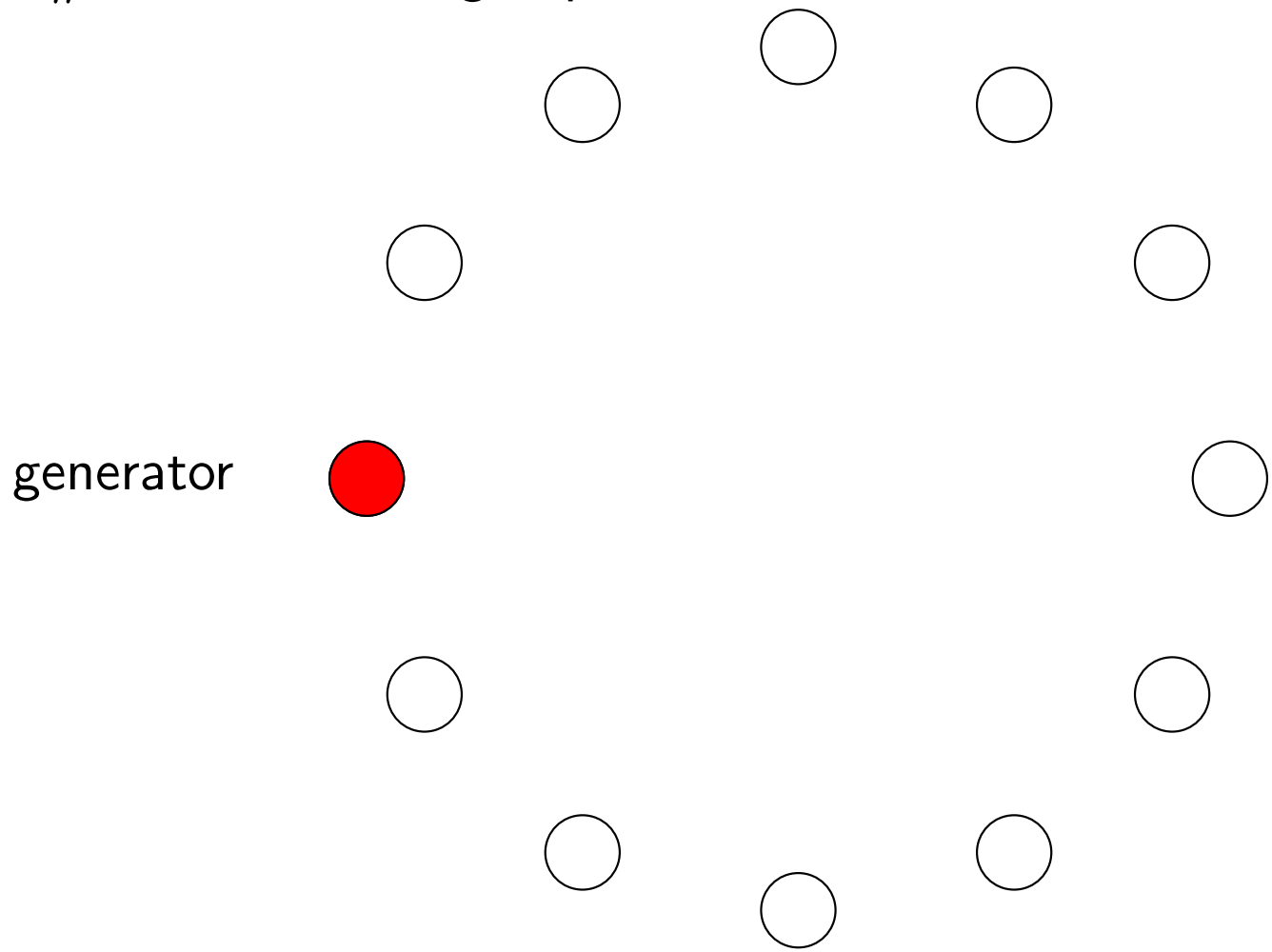
generator



Reminder: groups, subgroups, and generators

Small subgroup

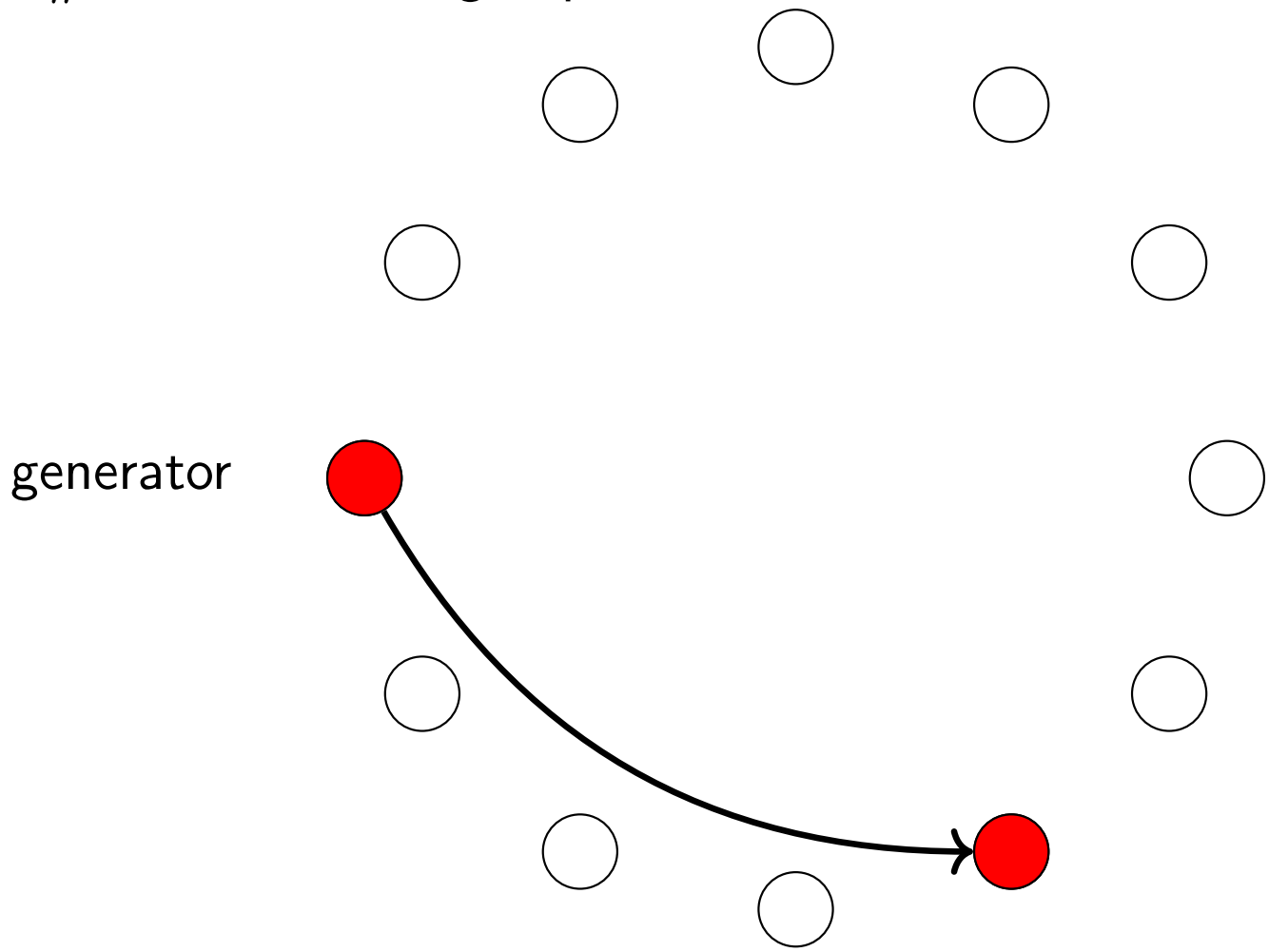
Order = #elements in subgroup



Reminder: groups, subgroups, and generators

Small subgroup

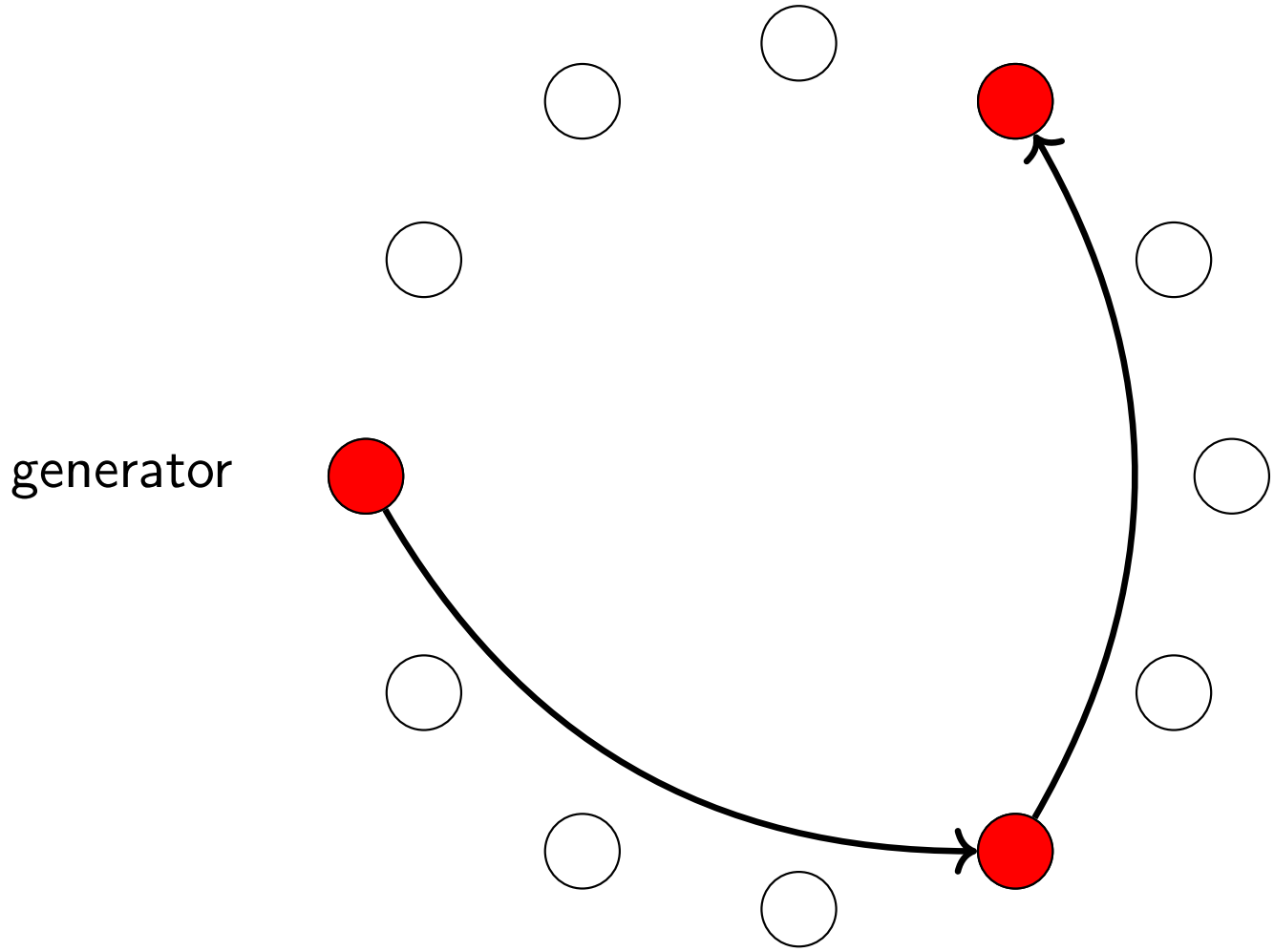
Order = #elements in subgroup



Reminder: groups, subgroups, and generators

Small subgroup

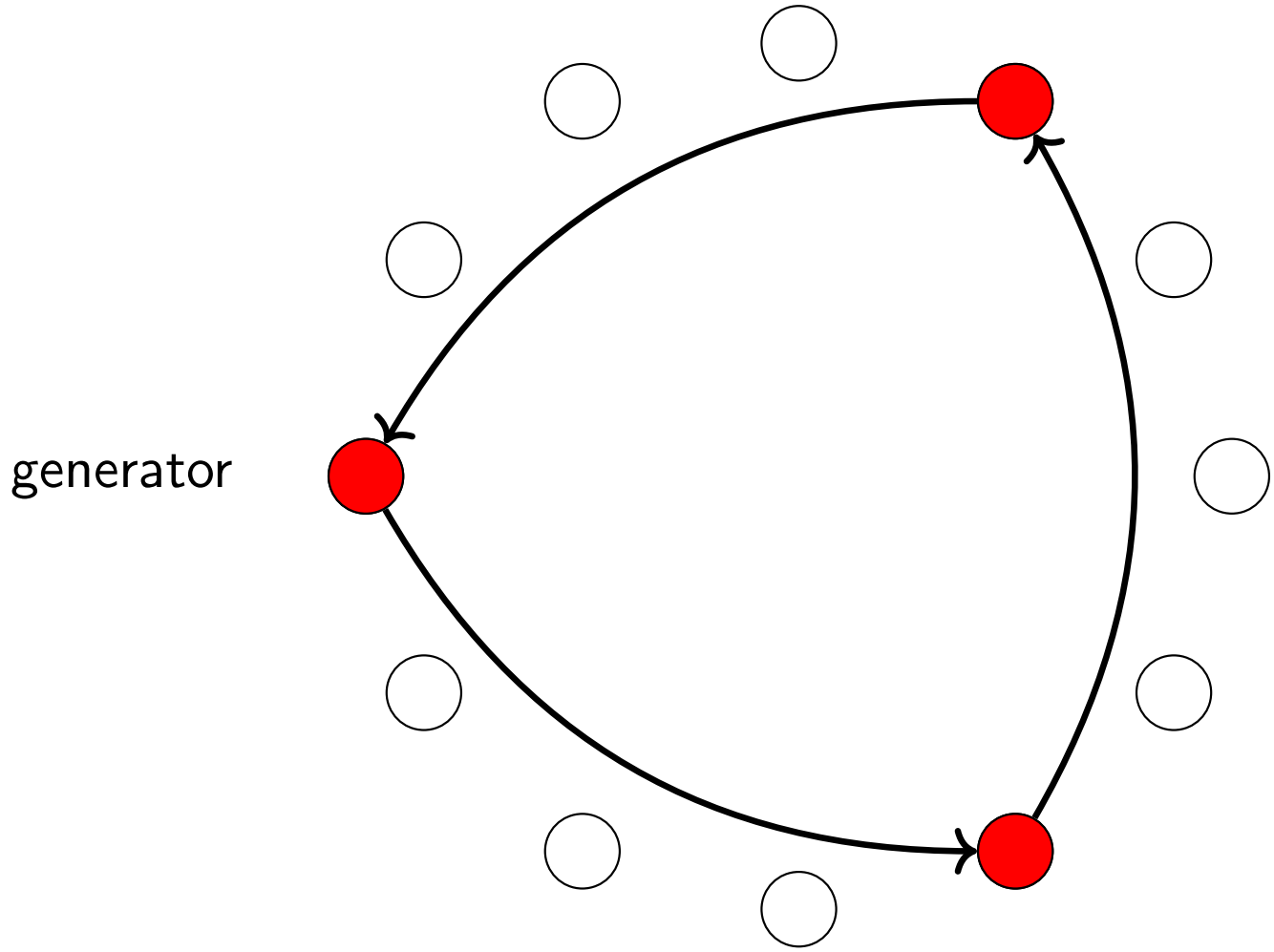
Order = #elements in subgroup



Reminder: groups, subgroups, and generators

Small subgroup

Order = #elements in subgroup



Elementary discrete log cryptanalysis

Theorem (Lagrange)

$\mathrm{ord}(g)_p$ *divides* $p-1$.

Theorem (Pollard rho, Shanks baby step giant step)

Discrete log takes $O(\sqrt{q})$ *time in (sub)group of order* q.

Pohlig-Hellman Algorithm

1. Factor group order $q = \prod_i q_i^{e_i}$.
2. Solve discrete log in each subgroup in time $e_i\sqrt{q_i}$.
3. Use Chinese remainder theorem to reconstruct log $\bmod q$.

A randomly chosen prime will have many small factors for $p-1$

Composite group orders: A sad tale

[ABDGGHHSTVVWZZ 2015] following [van Oorschot Wiener 1996]

1. 3.4M HTTPS servers supported Diffie-Hellman in 2015
2. 70,000 distinct primes p
3. 4,800 primes where $(p-1)/2$ is not prime.
4. For 750 groups, learned prime factors of $\mathrm{ord}_p(g)$ (by opportunustically factoring $(p-1)/2$ with ECM).
   - Used in 40,000 connections across scans.

Surprising implementation choices

Modular exponentiation is expensive, so many implementations use short exponents:

- 128 or 160 bits with 1024-bit p.

```
p = random_prime(2**1024)
g = 2

a = random_integer(2**160)
y = modexp(g,a,p)
```

5. Computed secret exponent for 159 exchanges and partial information in 460 exchanges.

Countermeasures against elementary discrete log attacks

The countermeasures are well known, and built into every DH standard:

- g should generate a group of large prime order q modulo p
- To maximize q, set $p = 2q + 1$ "safe" prime

In practice, the DDH assumption is false.

(Decisional Diffie-Hellman)

Of 70,000 Diffie-Hellman groups in use for HTTPS in 2015:

	$\mathrm{ord}_p(g)$ composite	$\mathrm{ord}_p(g)$ prime
p "safe"	**64,000**	**1250**
p non-"safe"	**750**	**4,000**

Question: Is this a real vulnerability? Probably not.

Small subgroups often used in Diffie-Hellman

Until 2018, NIST recommended small subgroups; now allows safe primes or small subgroups.

NIST SP800-56a: Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography

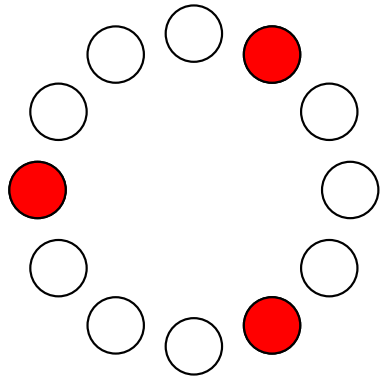
Table 1: FFC parameter-size sets

FFC parameter-size set name	FA	FB	FC
Maximum security strength supported (in bits)	80	112	112
Bit length of field size p (i.e., $\lceil \log_2 p \rceil$)	1024	2048	2048[1]
Bit length of subgroup order q (i.e., $\lceil \log_2 q \rceil$)	160	224	256

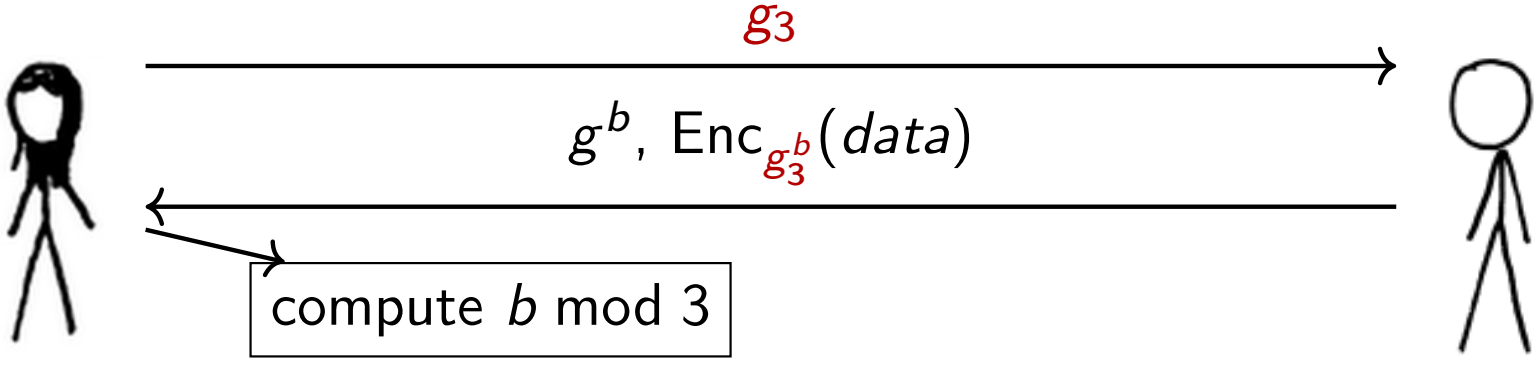
Existence of small subgroups $\rightarrow$ small subgroup attacks.

g generates correct subgroup of order q

g_3 generates subgroup of order 3



[Lim Lee 1997]



Well-known countermeasure: Validate group order.

1. Verify $2 \leq y \leq p - 2$.
2. Verify $1 = y^q \bmod p$.

Well-known countermeasure: Validate group order.

1. Verify $2 \leq y \leq p - 2$.
2. Verify $1 = y^q \bmod p$.

- I can't. Many protocols have no way to specify q.
- I don't want to. It is unnecessary.

Implementations don't validate group order.

[VASCFHHH 2017]

				Hosts accepting...			
	Hosts	**DHE**	**Non-Safe Primes**	0	1	-1	g_3/g_7
HTTPS	40M	30%	14%	0.6%	3%	5%	$\approx$ 100%
IKEv1	2.6M	100%	13%	*	28%	27%	99%
IKEv2	1.3M	100%	14%	*	0%	0%	97%
SSH	15M	71%	$\approx$ 0%	3%	25%	33%	N/A

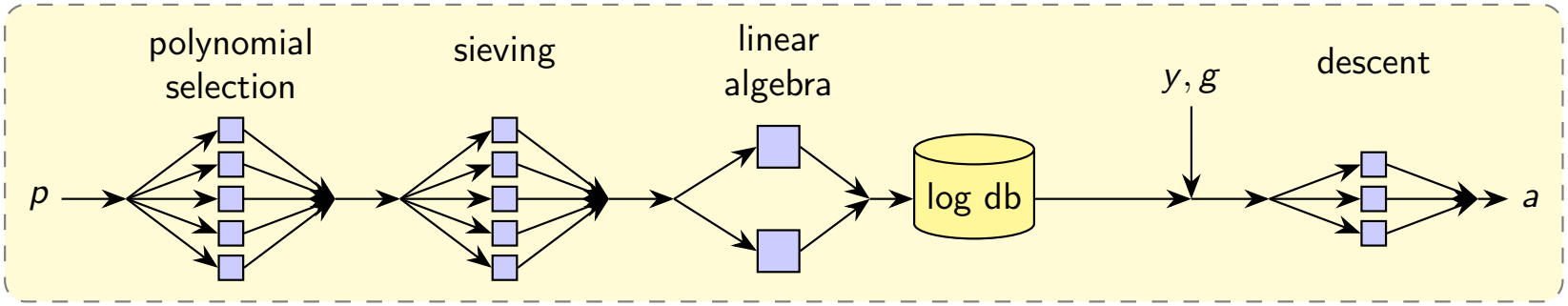
- OpenSSL vulnerable to full Lim-Lee exponent recovery attack for RFC 5114 primes.
- Amazon Load Balancer partial exponent recovery attack.

*: Did not scan: 0 causes unpatched Libre/Openswan to restart IKE daemon.

Computing discrete logs the hard way.

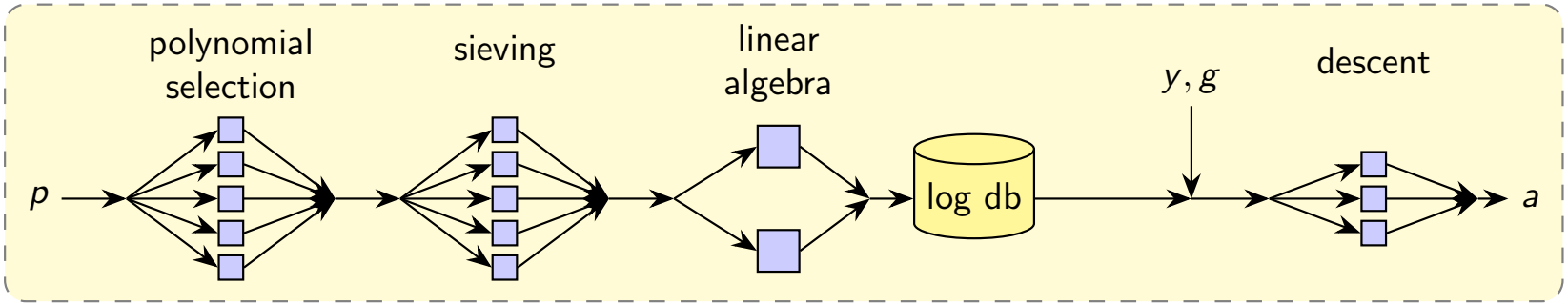
Number field sieve discrete log algorithm

[Gordon], [Joux, Lercier], [Semaev]



1. **Polynomial selection**: Find a good choice of number field K.
2. **Relation collection**: Factor elements over $\mathcal{O}_K$ and over $\mathbb{Z}$.
3. **Linear algebra**: Once there are enough relations, solve for logs of small elements.
4. **Individual log**: "Descent" Try to write target t as sum of logs in known database.

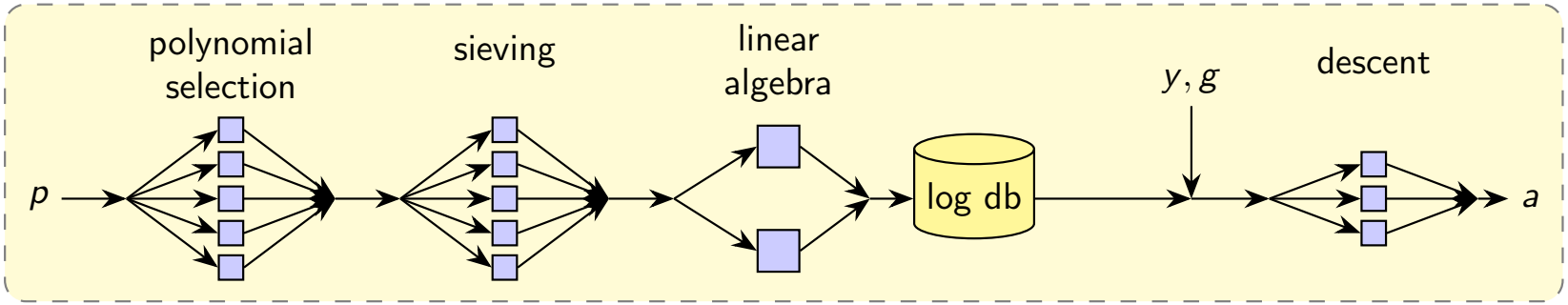
How long does it take to compute discrete logs?



Answer 1: Asymptotic complexity.

$L_p(1/3, 1.923) = \exp(1.923(\log p)^{1/3}(\log\log p)^{2/3})$

How long does it take to compute discrete logs?



Answer 1: Asymptotic complexity.

$L_p(1/3, 1.923) = \exp(1.923(\log p)^{1/3}(\log\log p)^{2/3})$

Answer 2: Computational records.

Year	Discrete Log	Factoring	
1999		512 bits	[Cavallar et al]
2005	431 bits		[Joux, Lercier]
2007	530 bits		[Kleinjung]
2009		768 bits	[Kleinjung et al.]
2014	596 bits		[Bouvier et al.]
2016	768 bits		[Kleinjung et al.]

"Perfect Forward Secrecy"

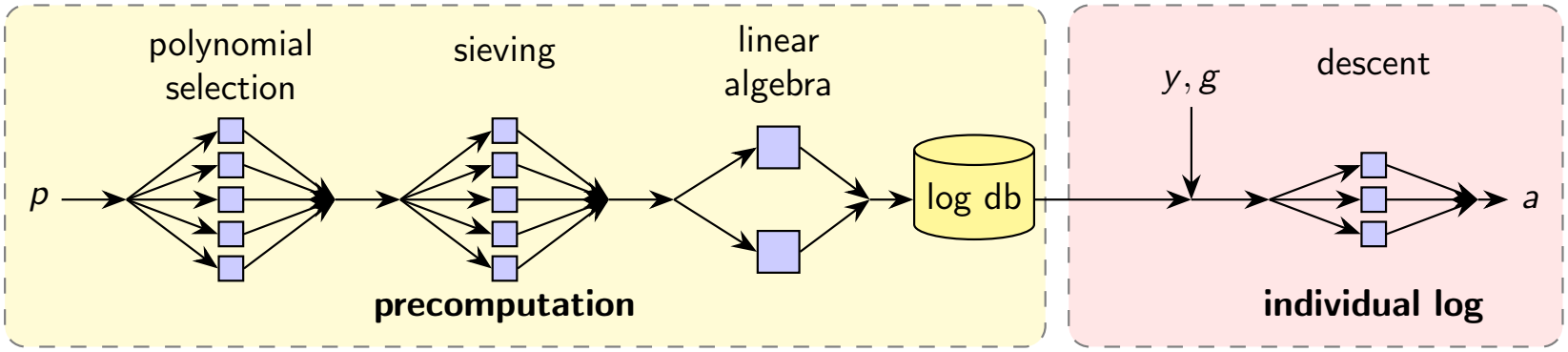
"Sites that use perfect forward secrecy can provide better security to users in cases where the encrypted data is being monitored and recorded by a third party."

"With Perfect Forward Secrecy, anyone possessing the private key and a wiretap of Internet activity **can decrypt nothing**."

"Ideally the DH group would match or exceed the RSA key size but **1024-bit DHE is arguably better than straight 2048-bit RSA** so you can get away with that if you want to."

"But in practical terms the risk of private key theft, for a non-ephemeral key, dwarfs out any cryptanalytic risk for any RSA or DH of 1024 bits or more; in that sense, PFS is a must-have and **DHE with a 1024-bit DH key is much safer than RSA-based cipher suites**, regardless of the RSA key size."

How long does it take to compute discrete logs?

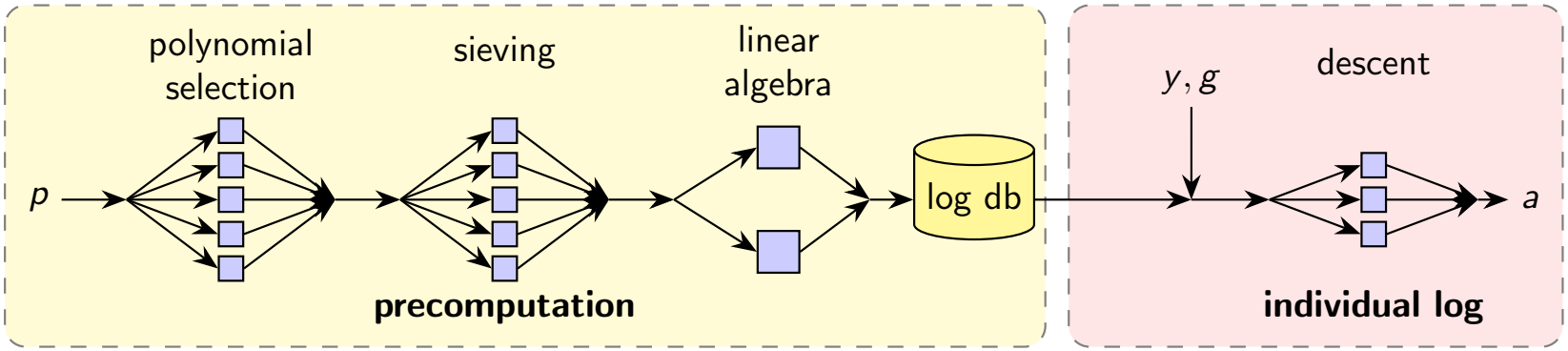


Answer 1.5:

$L_p(1/3, 1.923) = \exp(1.923(\log p)^{1/3}(\log\log p)^{2/3})$ $\qquad$ $L_p(1/3, 1.232)$

Precomputation can be done once and reused for many individual logs!

How long does it take to compute discrete logs?



Answer 3: Extrapolation.

	Precomputation core-years	Individual Log core-time
RSA-512	1	—
DH-512	10	10 mins
RSA-768	1,000	—
DH-768	5,000	4 days
RSA-1024	1,000,000	—
DH-1024	≈10,000,000	30 days

Implications for Diffie-Hellman security in practice

- 2015: **Logjam attack** [ABDGGHHSTVVW 2015]
  Modern TLS connections can be downgraded to 512-bit export-grade DH. 8% of popular HTTPS sites vulnerable.

Implications for Diffie-Hellman security in practice

- 2015: **Logjam attack** [ABDGGHHSTVVW 2015]
  Modern TLS connections can be downgraded to 512-bit export-grade DH. 8% of popular HTTPS sites vulnerable.

- Mass surveillance:
  Governments can exploit 1024-bit discrete log for wide-scale passive decryption.

Is breaking 1024-bit DH within reach of governments?

[ABDGGHHSTVVWZZ 2015]

	Precomputation core-years	Individual Log core-time
RSA-512	1	—
DH-512	10	10 mins
RSA-768	1,000	—
DH-768	5,000	4 days
RSA-1024	1,000,000	—
DH-1024	$\approx$10,000,000	30 days

Is breaking 1024-bit DH within reach of governments?

[ABDGGHHSTVVWZZ 2015]

	Precomputation core-years	Individual Log core-time
RSA-512	1	—
DH-512	10	10 mins
RSA-768	1,000	—
DH-768	5,000	4 days
RSA-1024	1,000,000	—
DH-1024	≈10,000,000	30 days

- With special-purpose hardware, 1024-bit DL feasible for ≈$100Ms.
- Then, individual logs can be computed in close to real time

James Bamford, 2012, Wired

According to another top official also involved with the program, the NSA made an enormous breakthrough several years ago in its ability to cryptanalyze, or break, unfathomably complex encryption systems employed by not only governments around the world but also many average computer users in the US. The upshot, according to this official: "Everybody's a target; everybody with communication is a target."

[...]

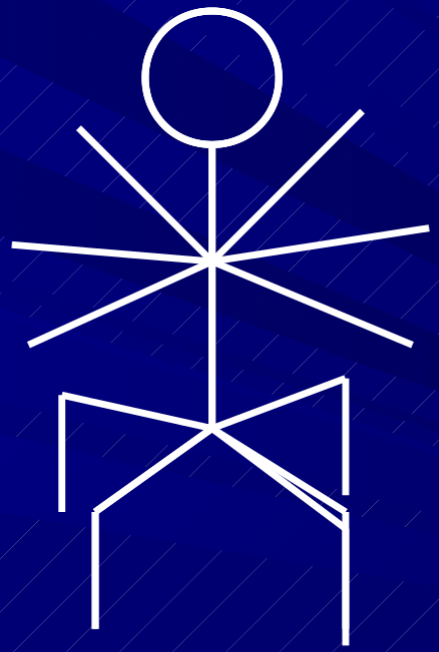
The breakthrough was enormous, says the former official, and soon afterward the agency pulled the shade down tight on the project, even within the intelligence community and Congress. "Only the chairman and vice chairman and the two staff directors of each intelligence committee were told about it," he says. The reason? "They were thinking that this computing breakthrough was going to give them the ability to crack current public encryption."

4. Communicate Results

Can we decrypt the VPN traffic?

- If the answer is “No” then explain how to turn it into a “YES!”
- If the answer is “YES!” then…

Happy Dance!!



Possible explanations for passive decryption capabilities

- Discrete logs in Oakley Group 2
  - Precomputation for a single 1024-bit prime allows passive decryption of connections to 66% of VPN servers and 26% of SSH servers.
- Backdoored RNGs
  - Example: Dual-EC DRBG [CMGFCGHWRS 2016]
- Flawed RNGs
  - ANSI X9.31 RNG [Cohney Green Heninger 2017]
- Custom implants
  - Example: Shadowbroker dump.

The good news

- TLS 1.3 made many good choices
  - Built-in downgrade and man-in-the-middle protection.
  - RSA key exchange removed.
  - Minimum 2048-bit prime DHE size; fixed nothing-up-my-sleeve groups.
- Minimum key strengths raised across the board in browsers.
- Non-EC Diffie-Hellman has been removed entirely from many browsers.
- Fraction of popular sites negotiating 1024-bit Diffie-Hellman decreased from 37% to 23% in year after our work.
- OpenSSH raised minimum key strengths and removed Oakley Group 2.

Mining your Ps and Qs: Widespread Weak Keys in Network Devices Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman *Usenix Security 2012*

A Messy State of the Union: Taming the Composite State Machines of TLS Benjamin Beurdouche, Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Markulf Kohlweiss, Alfredo Pironti, Pierre-Yves Strub, Jean Karim Zinzindohoue. *Oakland 2015*.

Imperfect Forward Secrecy: How Diffie-Hellman Fails in Practice Adrian, Bhargavan, Durumeric, Gaudry, Green, Halderman, Heninger, Springall, Thomé, Valenta, VanderSloot, Wustrow, Zanella-Béguelin, Zimmermann. *CCS 2015.* `weakdh.org`

Factoring as a Service Luke Valenta, Shaanan Cohney, Alex Liao, Joshua Fried, Satya Bodduluri, and Nadia Heninger. *FC 2016.* `seclab.upenn.edu/projects/faas/`

Weak keys remain widespread in network devices Marcella Hastings, Joshua Fried, and Nadia Heninger *IMC 2016*

Measuring small subgroup attacks against Diffie-Hellman. Luke Valenta, David Adrian, Antonio Sanso, Shaanan Cohney, Joshua Fried, Marcella Hastings, J. Alex Halderman, and Nadia Heninger. *NDSS 2017.*