On the Practical Exploitability of Dual EC in TLS Implementations

Stephen Checkoway,[1] Matthew Fredrikson,[2] Ruben Niederhagen,[3]
Adam Everspaugh,[2] Matthew Green,[1] Tanja Lange,[3]
Thomas Ristenpart,[2] Daniel J. Bernstein,[3,4]
Jake Maskiewicz,[5] and Hovav Shacham[5]

[1]JHU, [2]U. Wisconsin, [3]TU/e, [4]UIC, [5]UCSD

Dual EC DRBG (briefly)

- Pseudo random number generator (PRNG)
- ANSI/ISO/NIST standard designed by the NSA
- Shumow & Ferguson demonstrate potential backdoor in 2007
- Snowden reveals BULLRUN in 2013
- NSA paid RSA $10M to make Dual EC default PRNG

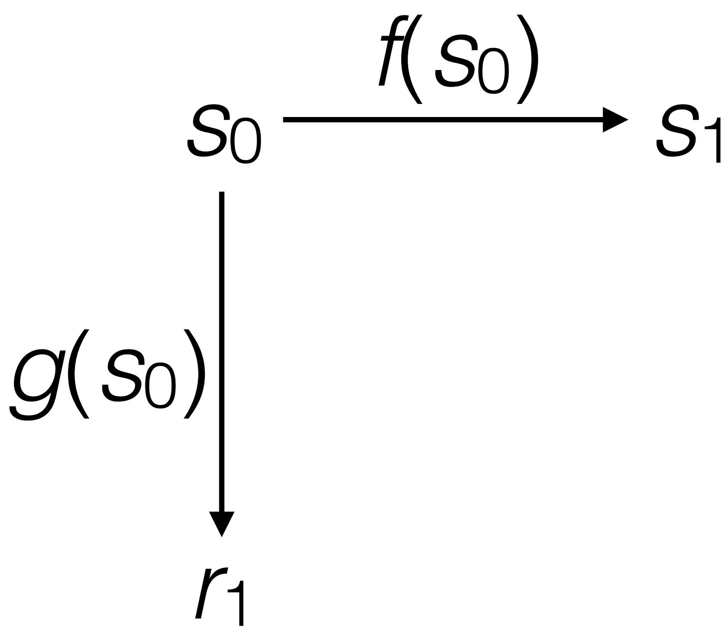
Our work

- YES: Difficulty of exploiting Dual EC backdoor in TLS implementations (assuming a backdoor)
- NO: Probability of a backdoor in Dual EC
- NO: Recovering the backdoor's secret key

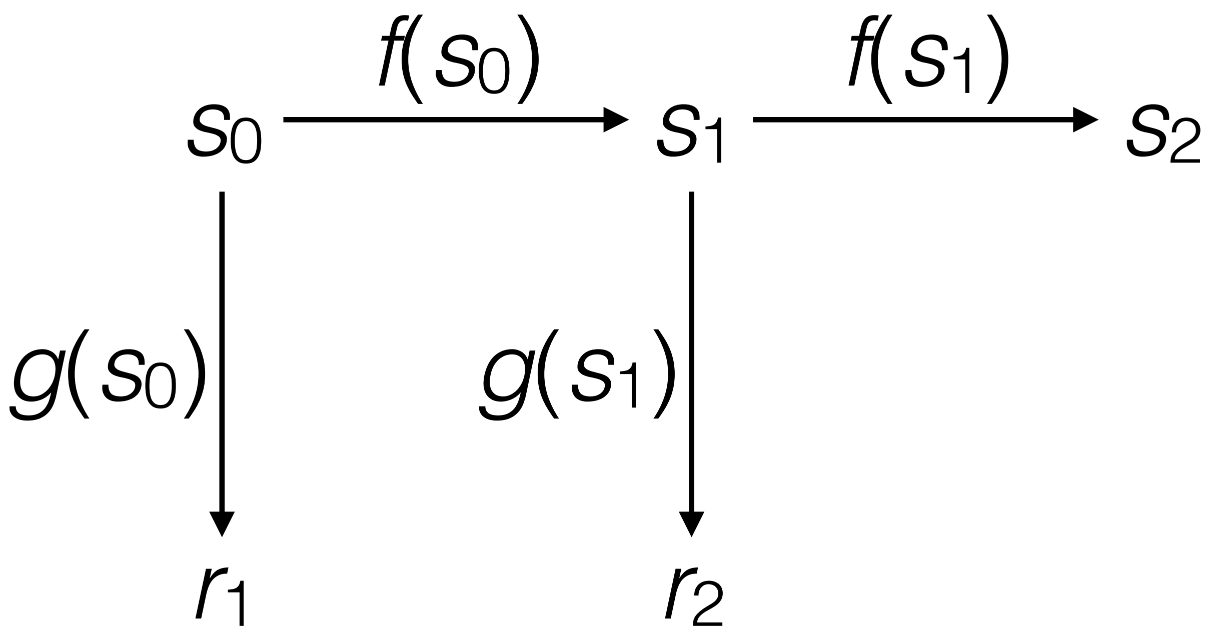
Pseudo random number generator (PRNG)

s_0

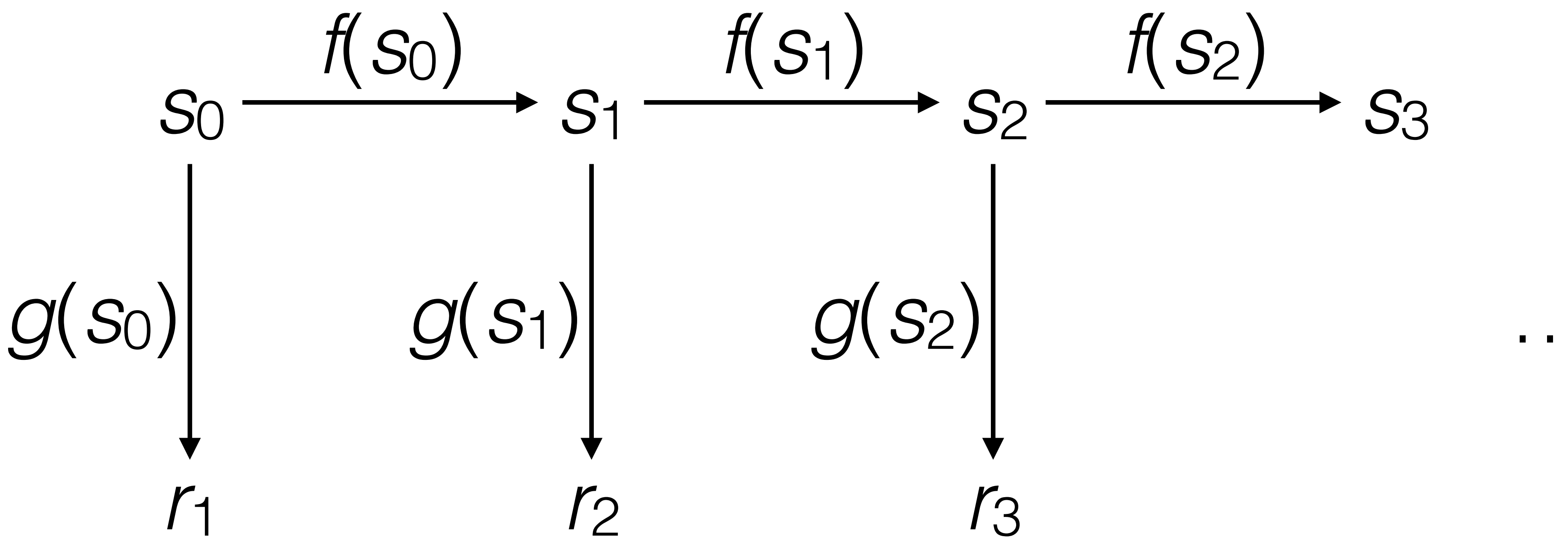
Pseudo random number generator (PRNG)



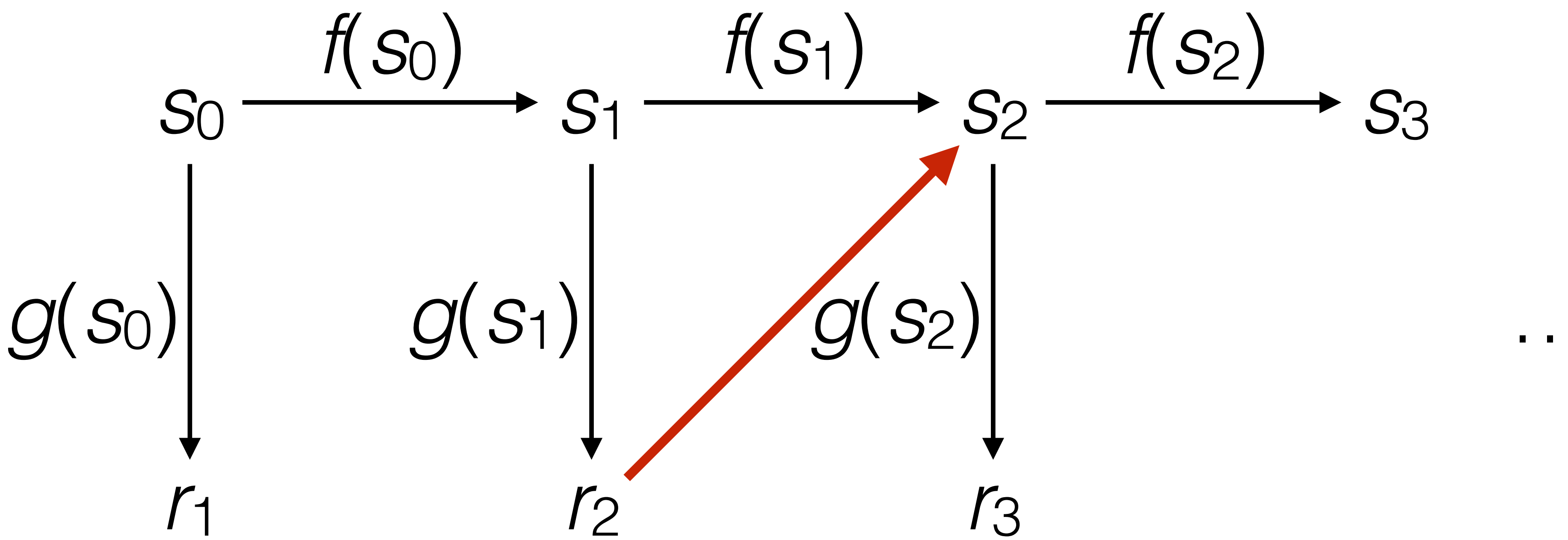
Pseudo random number generator (PRNG)



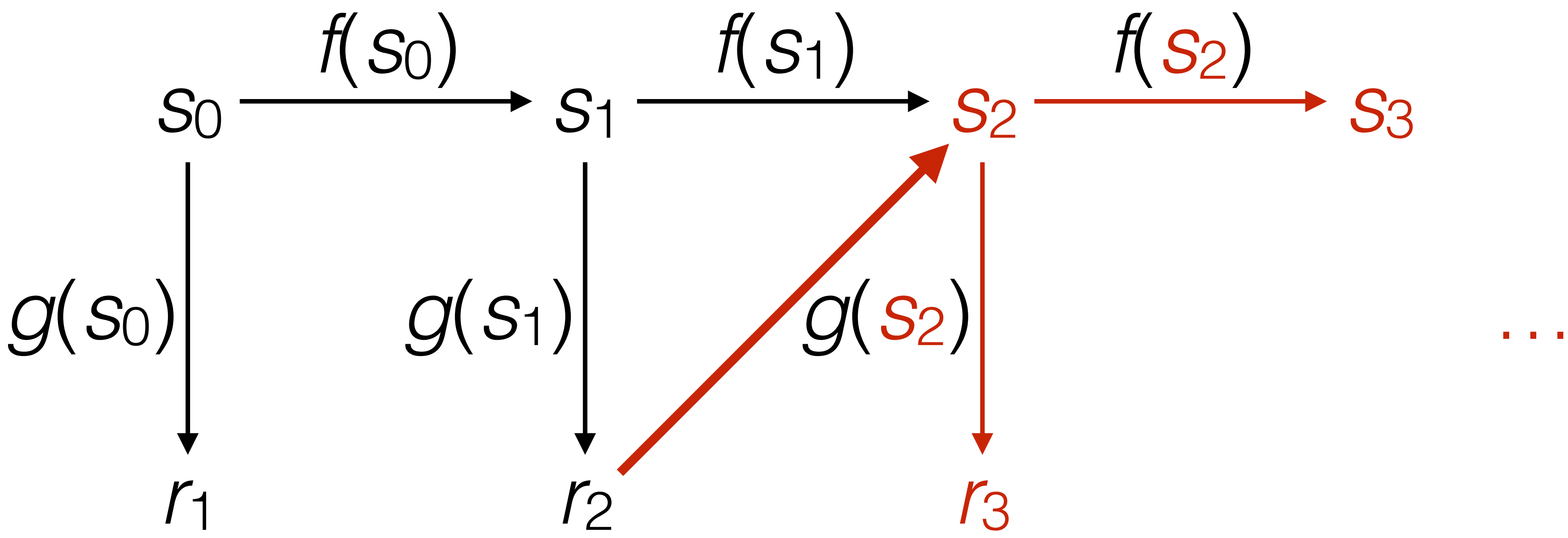
Pseudo random number generator (PRNG)



Pseudo random number generator (PRNG)



Pseudo random number generator (PRNG)



Elliptic curve primer

- Points on an elliptic curve are pairs (x, y)
- x and y are 32-byte integers
- Points can be added together to get another point
- Scalar multiplication: Given integer n and point P, $nP = P + P + \ldots + P$ is easy to compute
- Given points P and nP, n is hard to compute

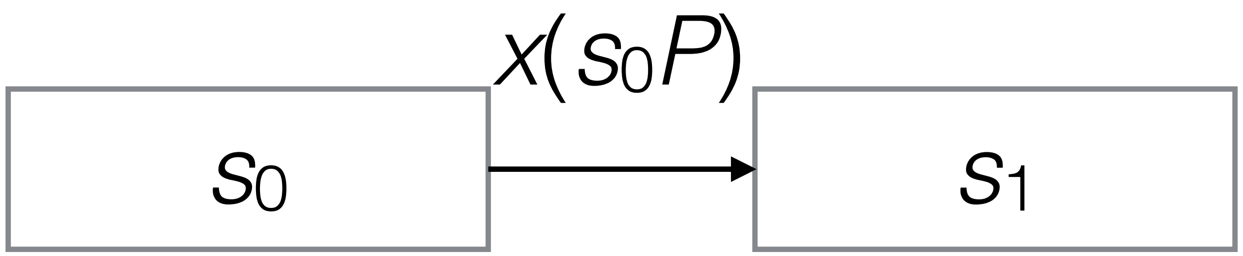
Dual EC operation (simplified)

s_0

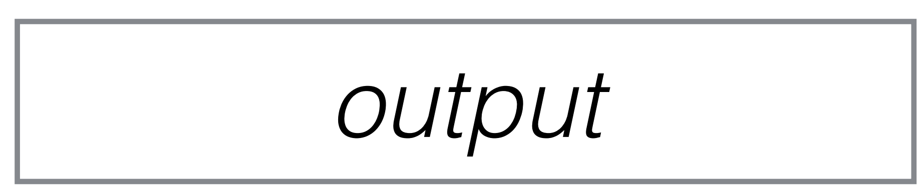
- 32-byte states

output

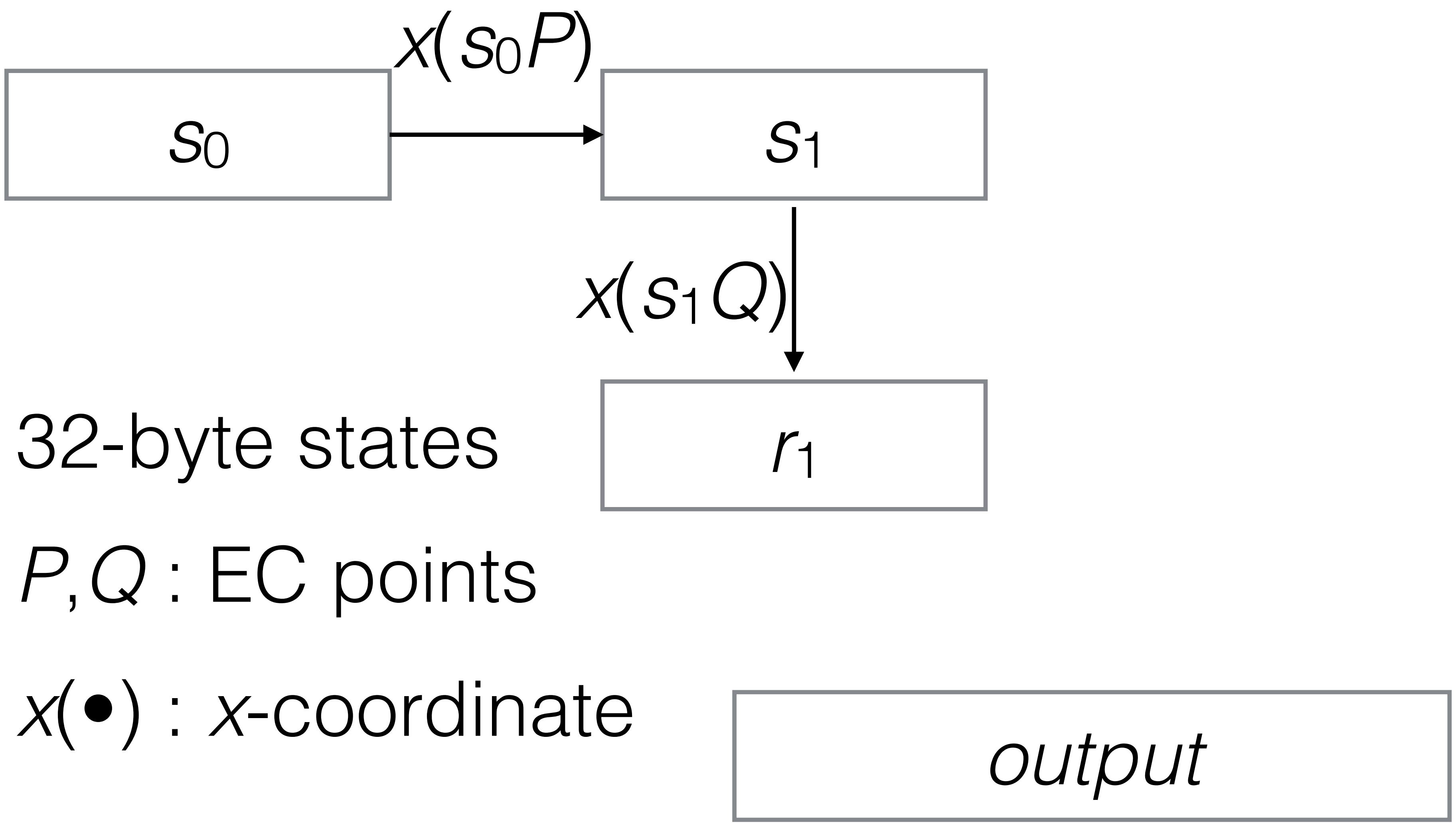
Dual EC operation (simplified)



- 32-byte states
- P, Q : EC points
- $x(\bullet)$: x-coordinate

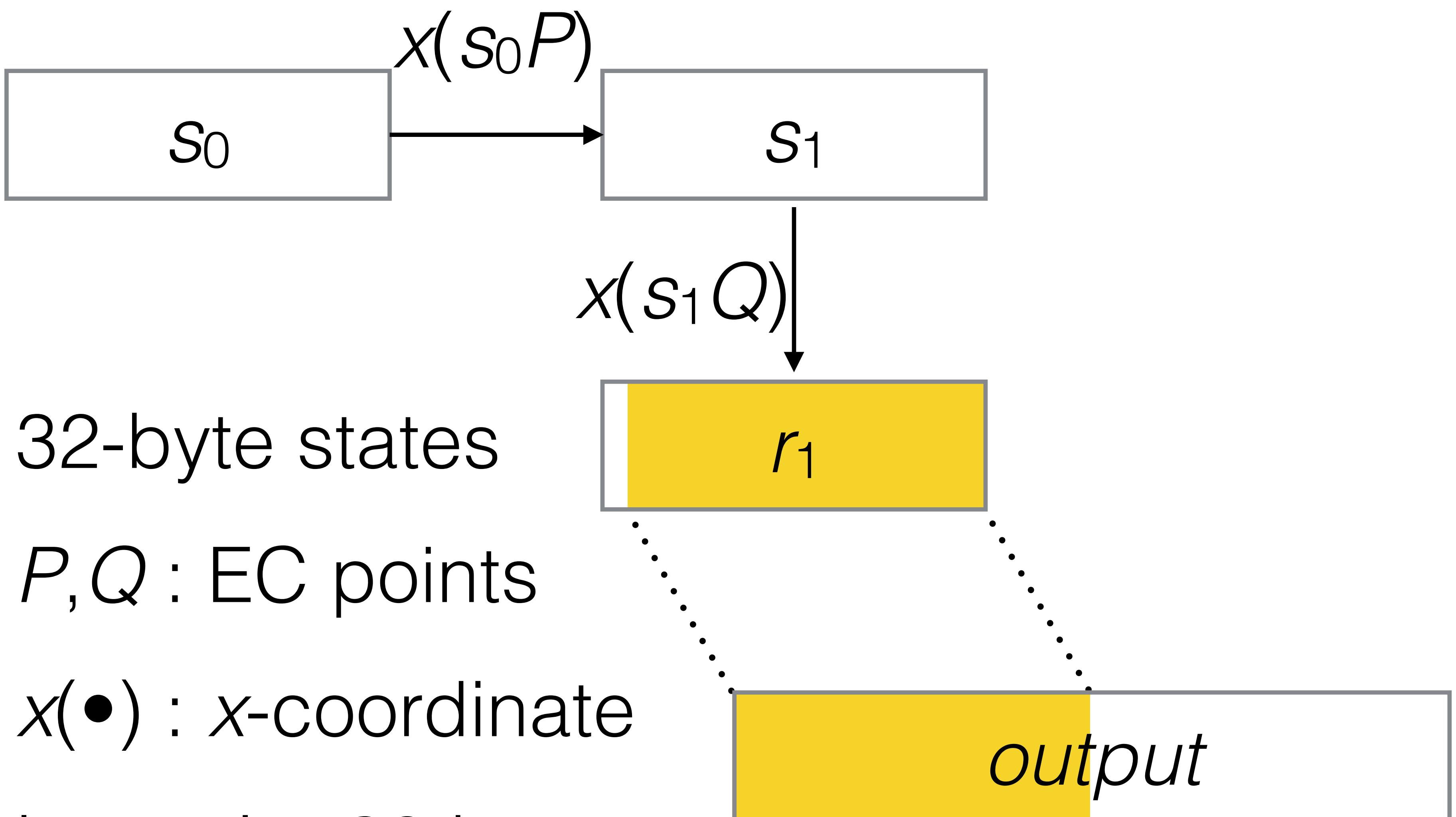


Dual EC operation (simplified)



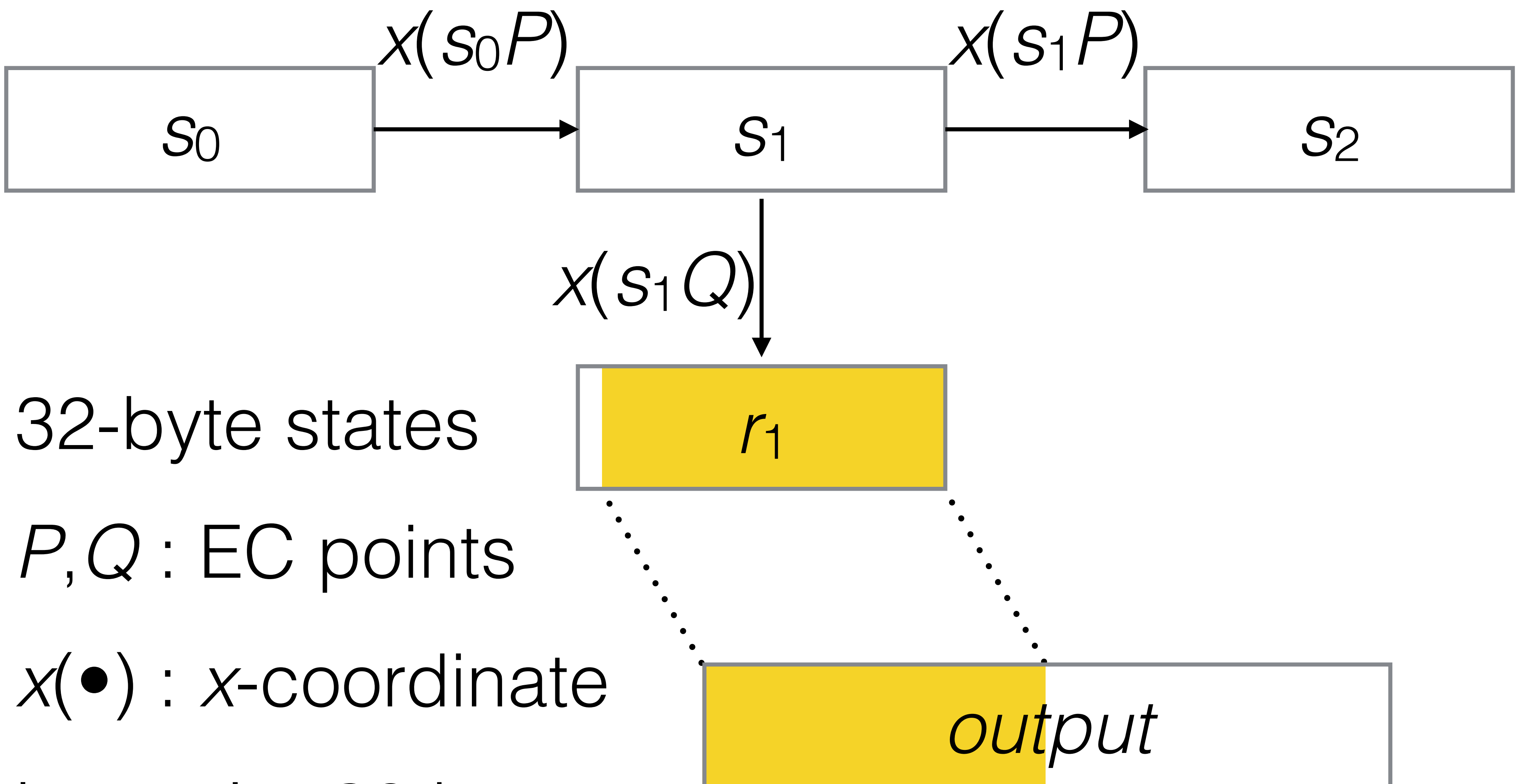
- 32-byte states
- P, Q : EC points
- $x(\bullet)$: x-coordinate

Dual EC operation (simplified)



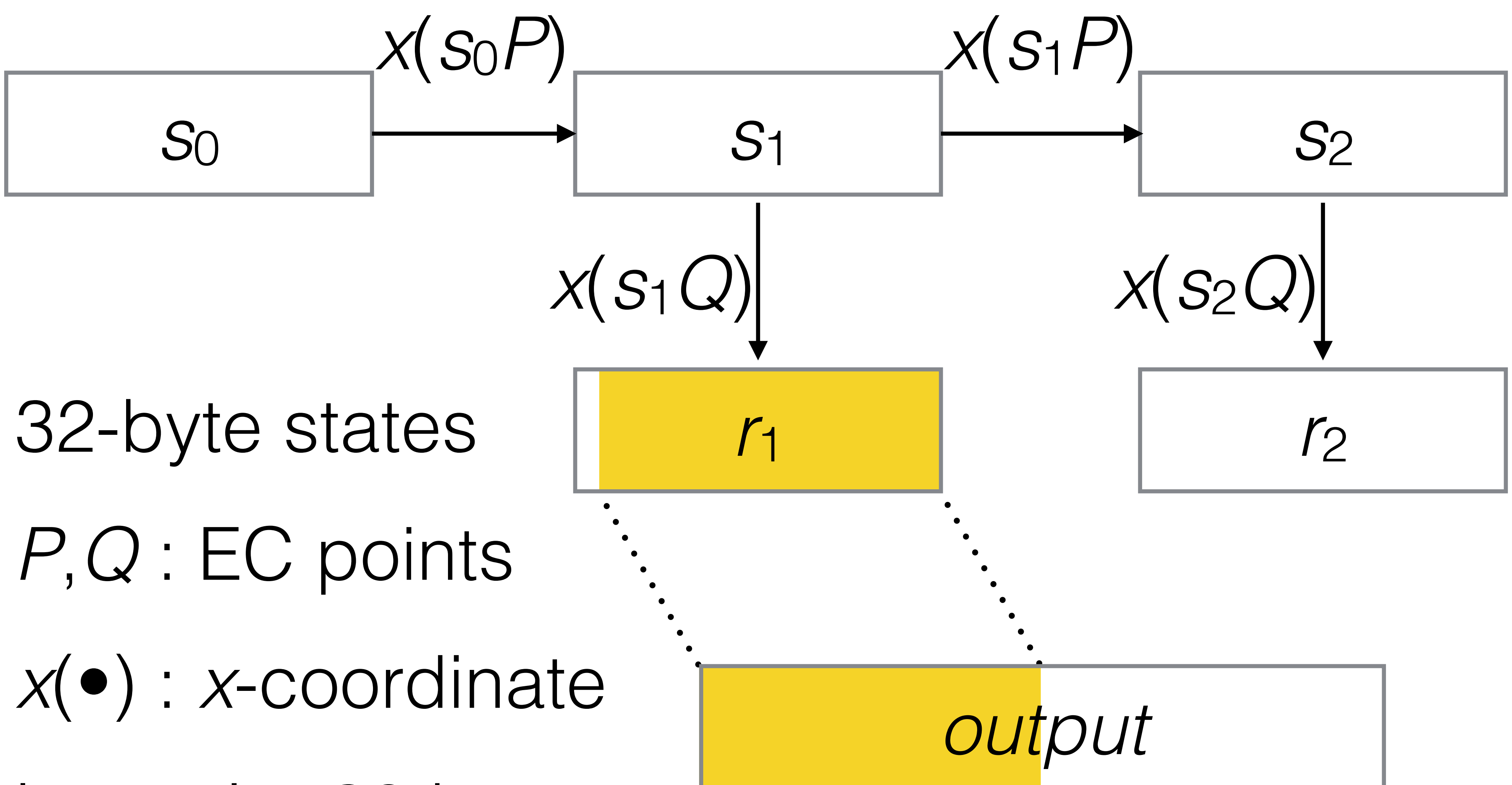
- 32-byte states
- P, Q : EC points
- $x(\bullet)$: x-coordinate
- least sig. 30 bytes of r_i form *output*

Dual EC operation (simplified)



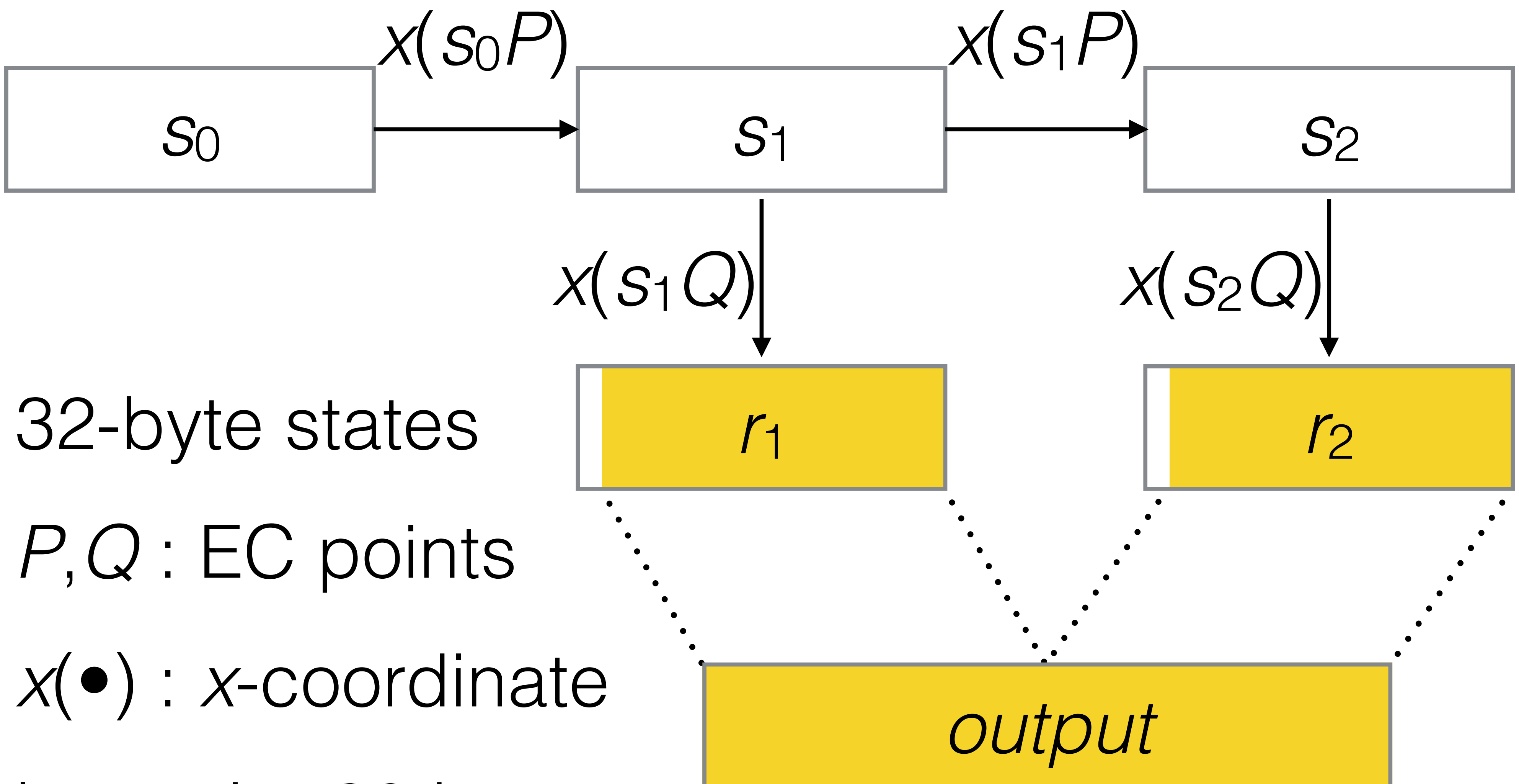
- 32-byte states
- P, Q : EC points
- $x(\bullet)$: x-coordinate
- least sig. 30 bytes of r_i form *output*

Dual EC operation (simplified)



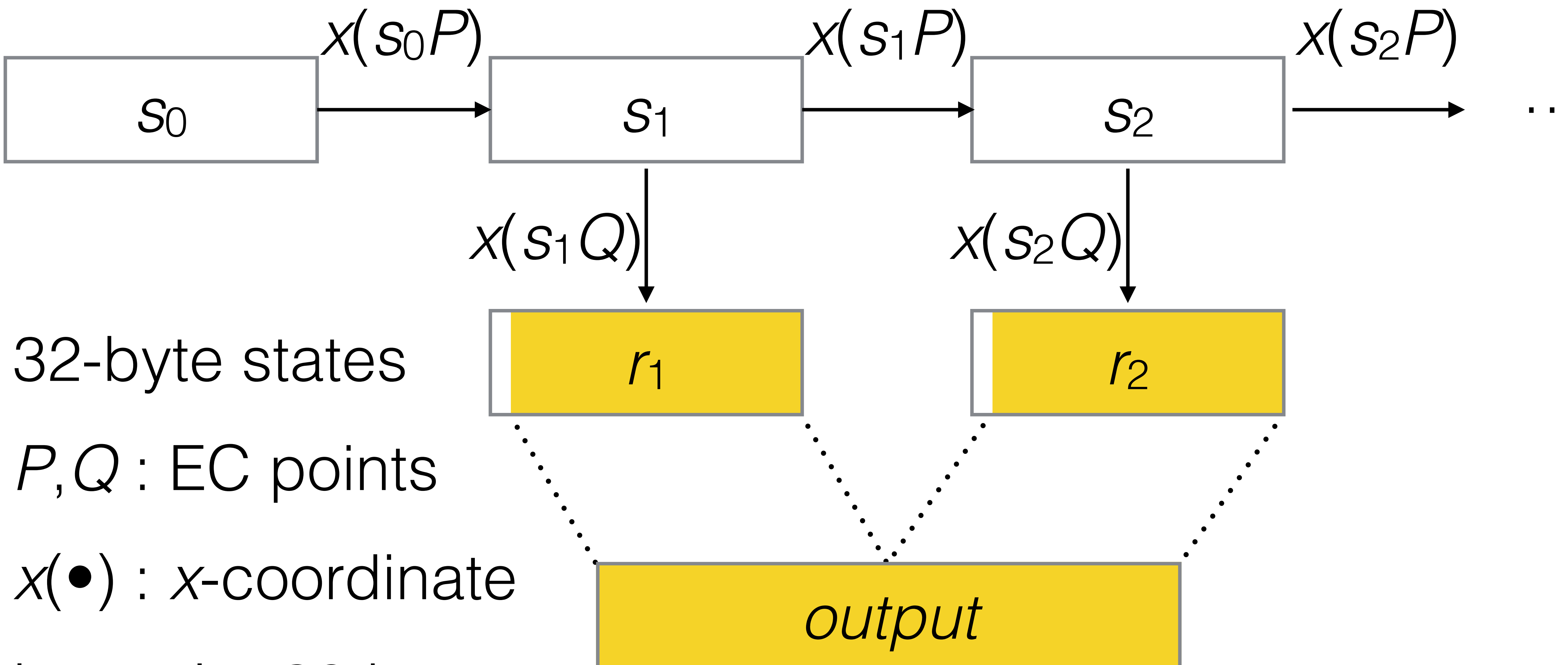
- 32-byte states
- P, Q : EC points
- $x(\bullet)$: x-coordinate
- least sig. 30 bytes of r_i form *output*

Dual EC operation (simplified)



- 32-byte states
- P, Q : EC points
- $x(\bullet)$: x-coordinate
- least sig. 30 bytes of r_i form *output*

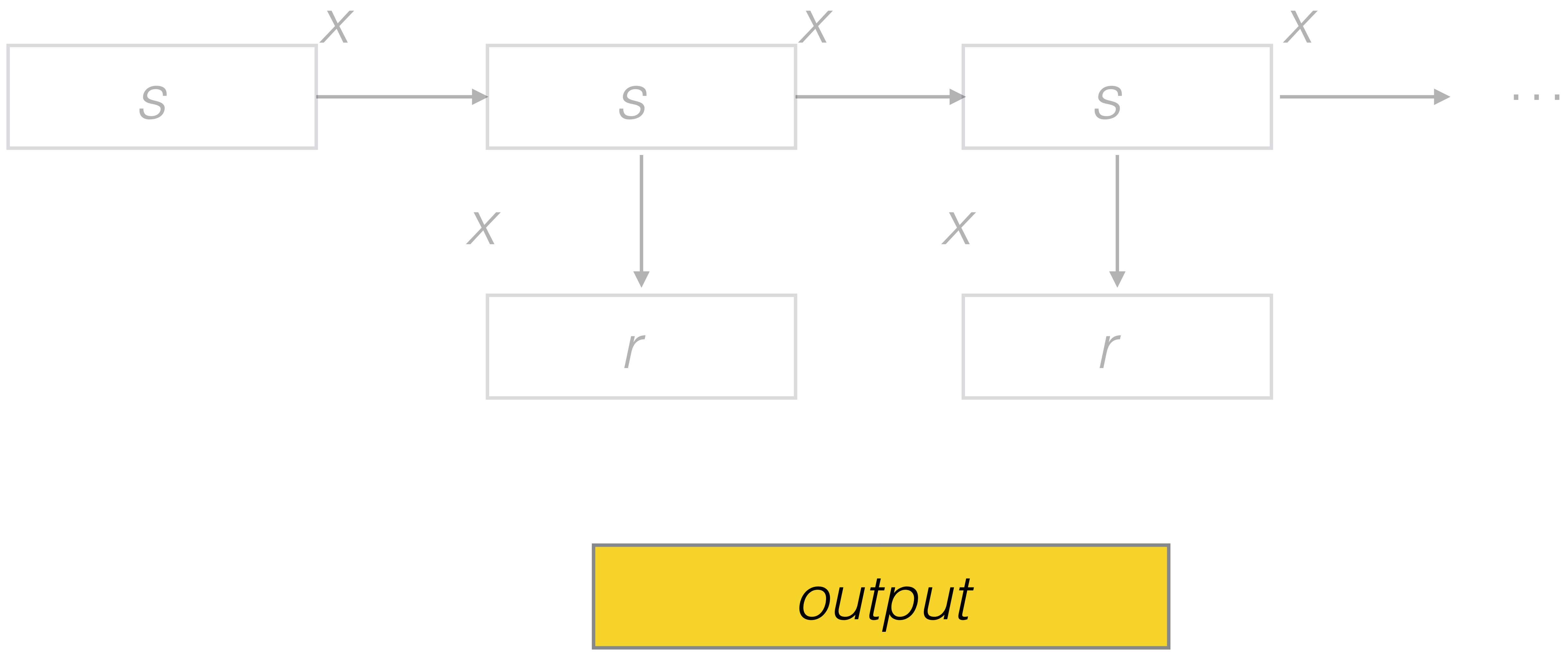
Dual EC operation (simplified)



- 32-byte states
- P, Q : EC points
- $x(\bullet)$: x-coordinate
- least sig. 30 bytes of r_i form *output*

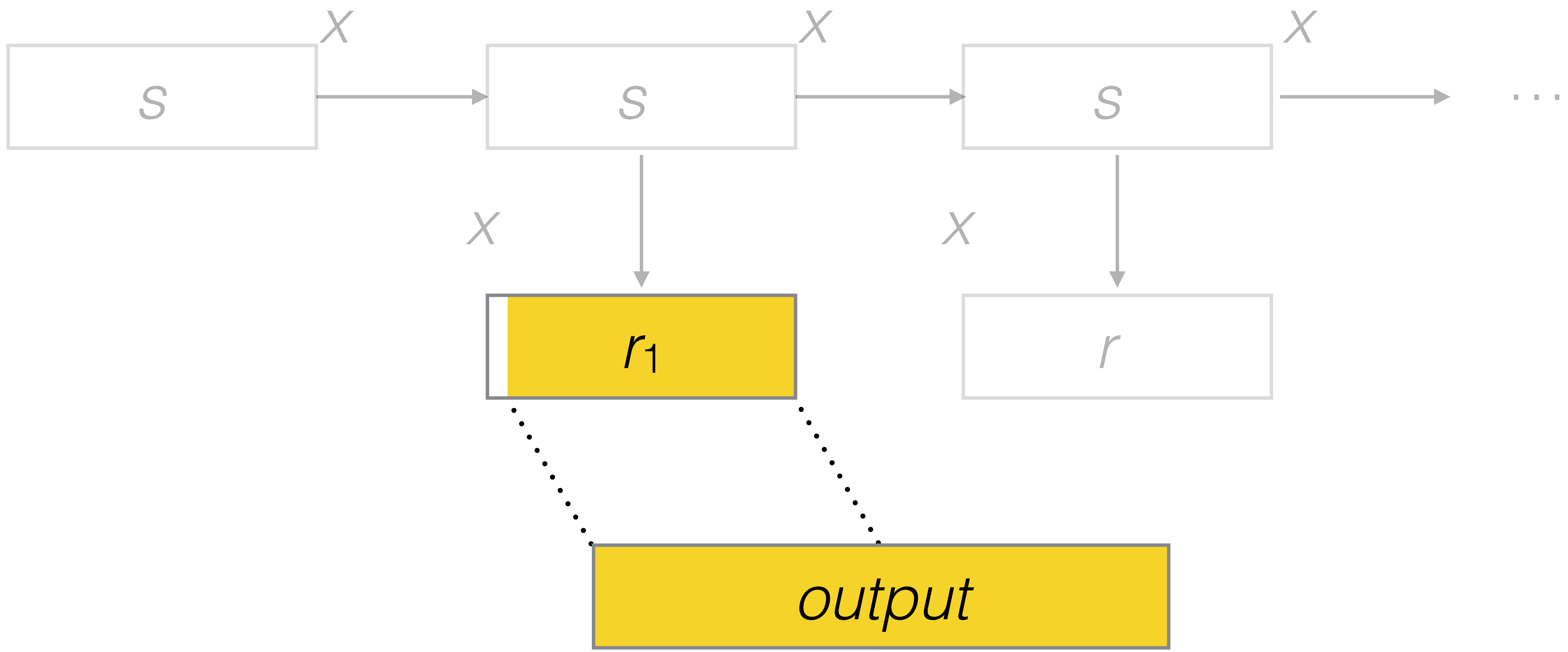
Shumow–Ferguson attack

Assumes known integer d s.t. $P = dQ$



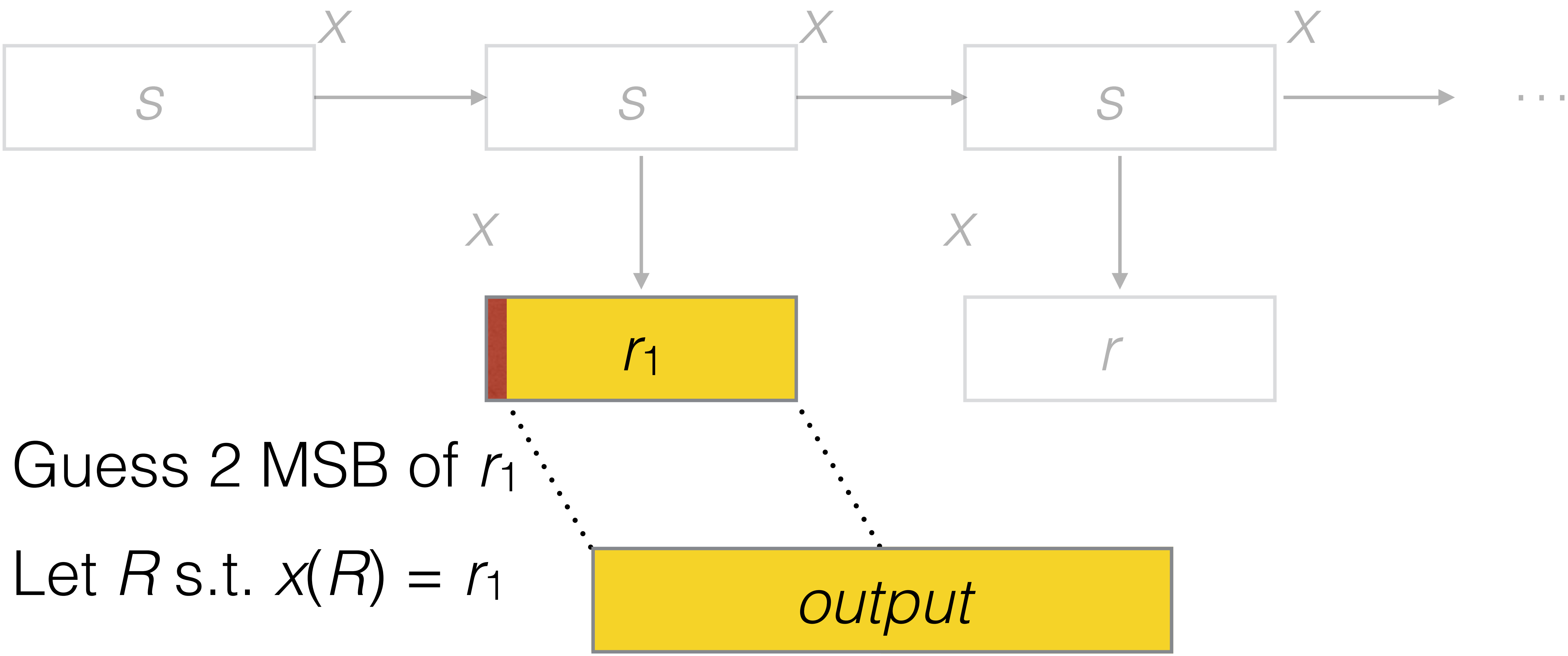
Shumow–Ferguson attack

Assumes known integer d s.t. $P = dQ$



Shumow–Ferguson attack

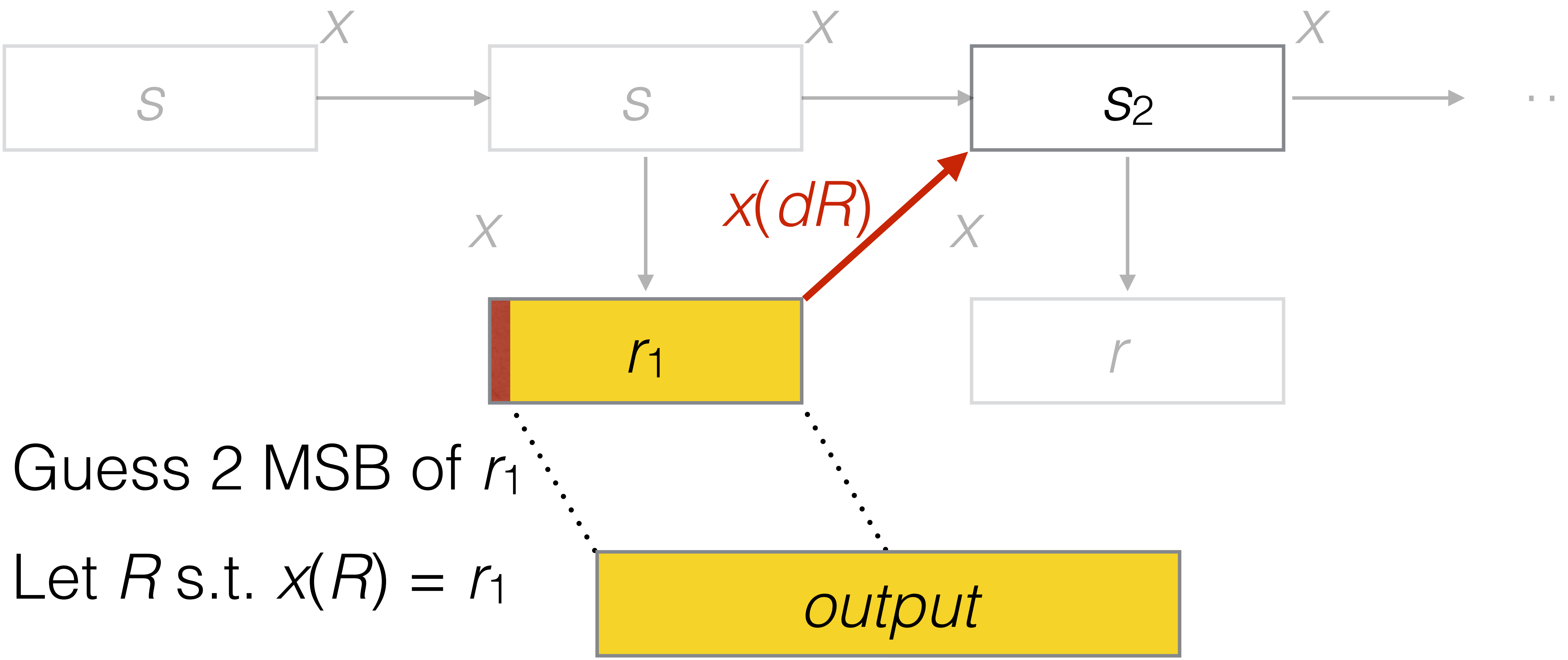
Assumes known integer d s.t. $P = dQ$



- Guess 2 MSB of r_1
- Let R s.t. $x(R) = r_1$

Shumow–Ferguson attack

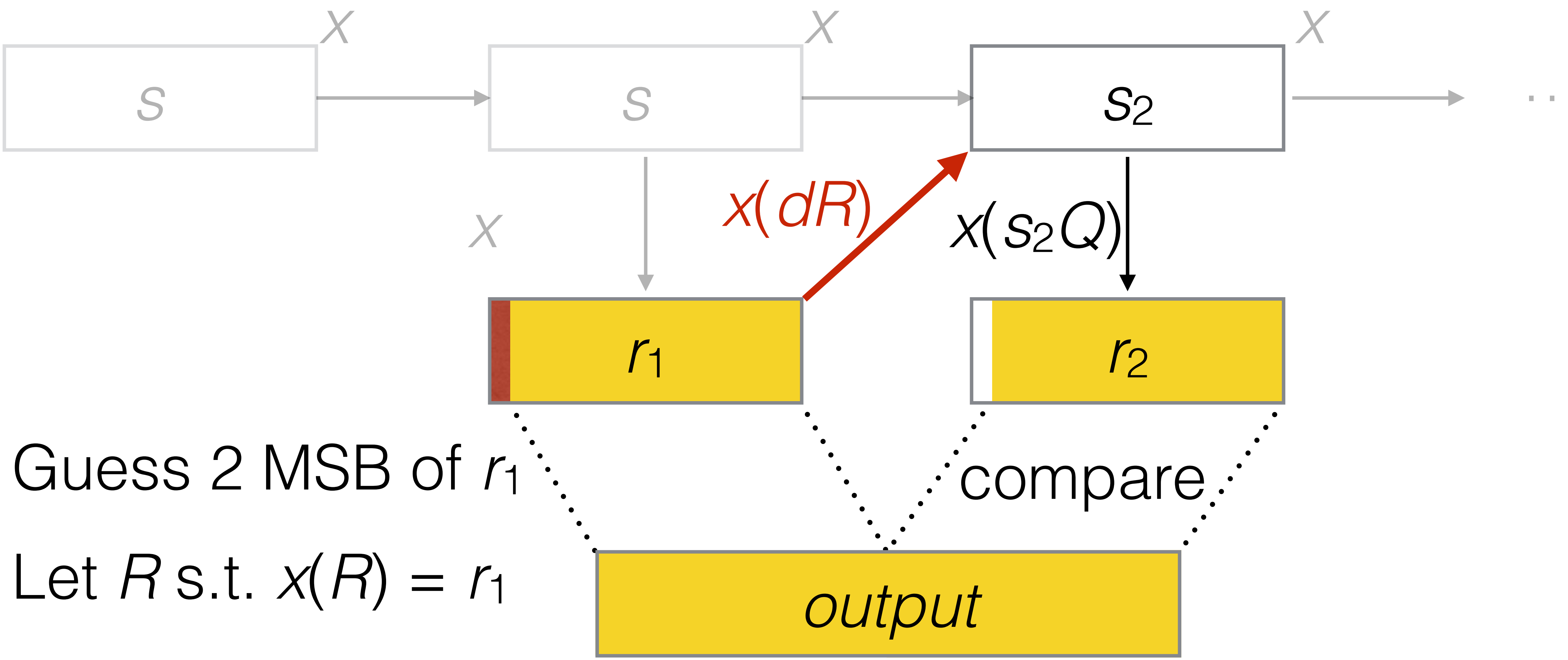
Assumes known integer d s.t. $P = dQ$



- Guess 2 MSB of r_1
- Let R s.t. $x(R) = r_1$
- Compute $s_2 = x(s_1P) = x(s_1dQ) = x(ds_1Q) = x(dR)$

Shumow–Ferguson attack

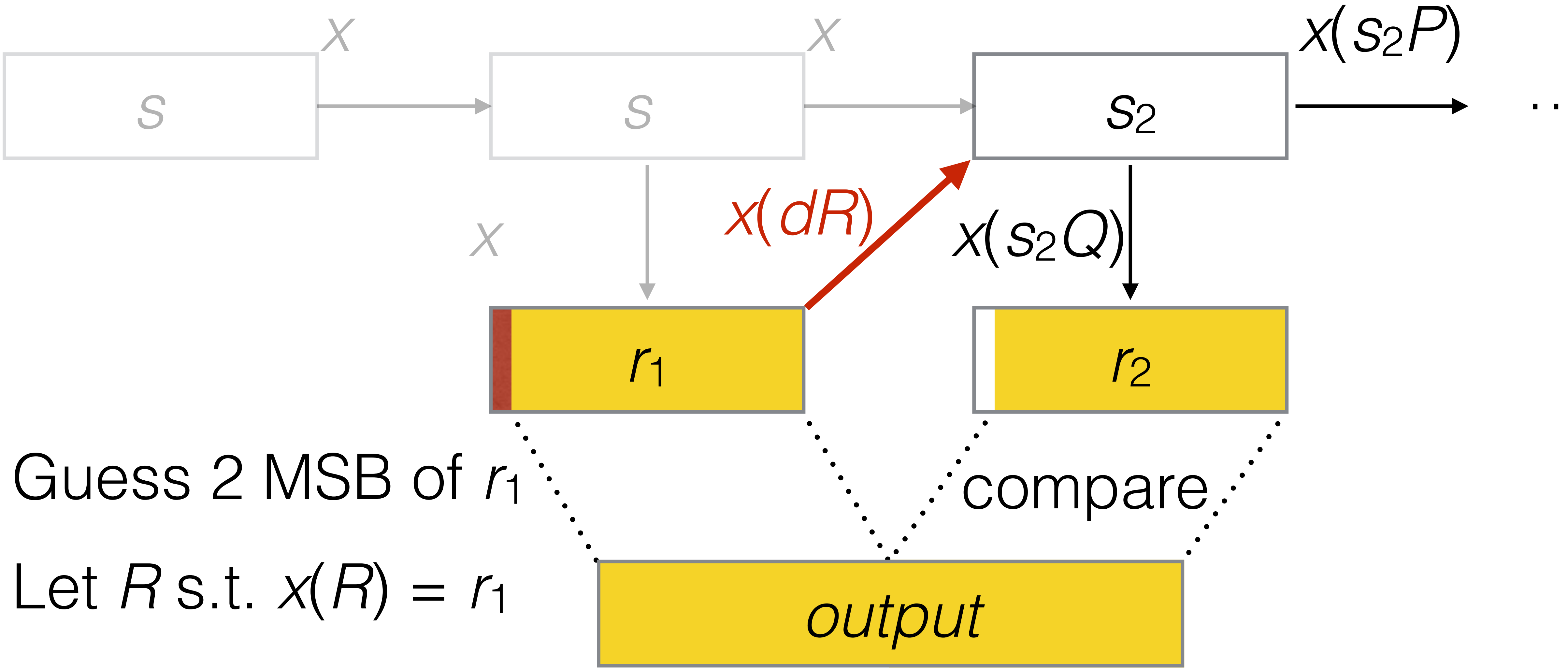
Assumes known integer d s.t. $P = dQ$



- Guess 2 MSB of r_1
- Let R s.t. $x(R) = r_1$
- Compute $s_2 = x(s_1P) = x(s_1dQ) = x(ds_1Q) = x(dR)$
- Compute r_2 and compare with *output*

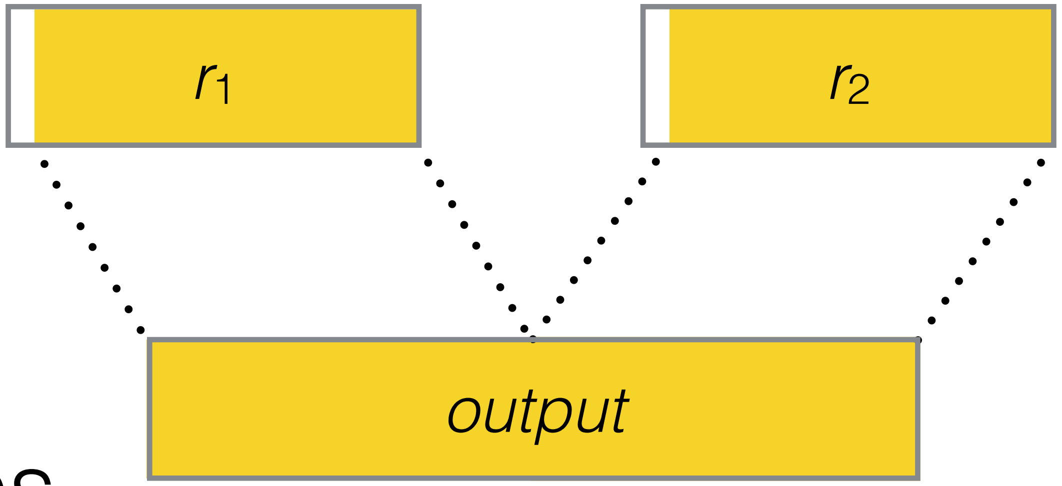
Shumow–Ferguson attack

Assumes known integer d s.t. $P = dQ$



- Guess 2 MSB of r_1
- Let R s.t. $x(R) = r_1$
- Compute $s_2 = x(s_1P) = x(s_1dQ) = x(ds_1Q) = x(dR)$
- Compute r_2 and compare with *output*

S–F attack prerequisites



- Attacker sees
  1. Most of r_1 (e.g., $\geq$ 28 bytes)
  2. Some public function of “enough” of r_2

TLS

client random →

server random, session id, xP ←

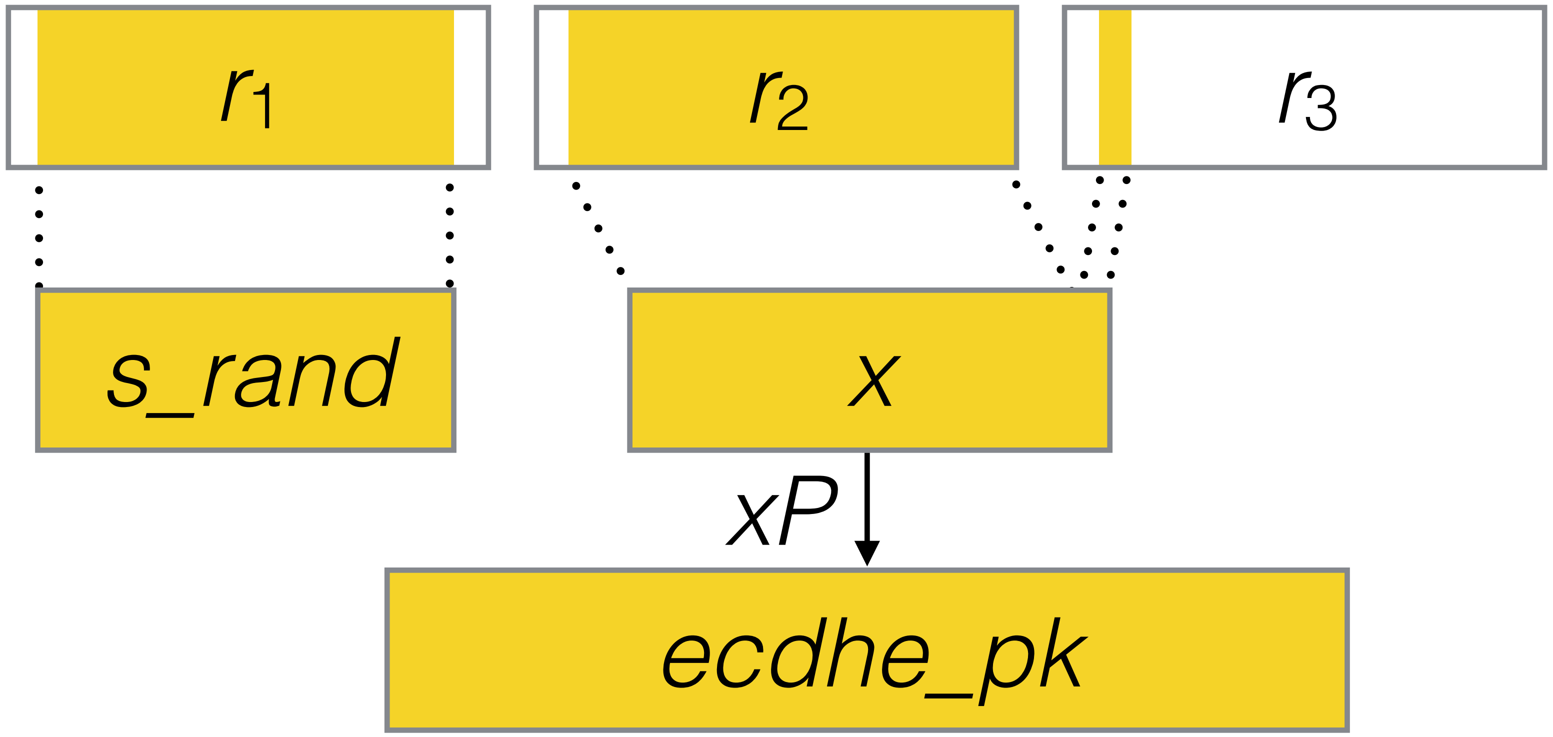
yP →

- {client,server} random: 28 random bytes
- session id: 32 bytes (can be random)
- x, y: ECDHE secret keys
- xP, yP: ECDHE public keys
- Recovering x or y enables decryption

Common TLS libraries

- RSA BSAFE Share for Java
- RSA BSAFE Share for C/C++
- Microsoft Secure Channel (SChannel)
- OpenSSL-FIPS (OpenSSL-Fixed)

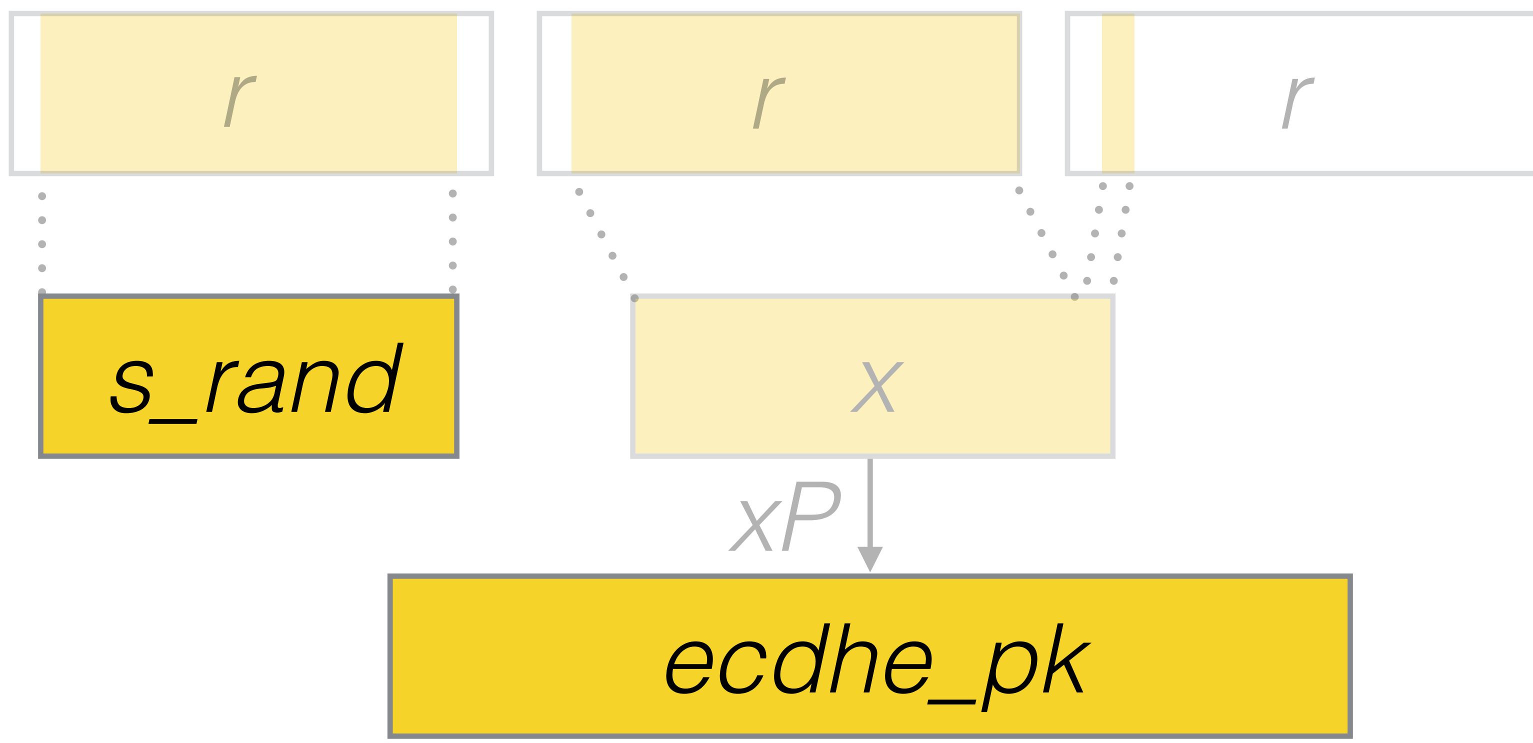
RSA BSAFE Share for Java



server_rand ← dual_ec(28)
x ← dual_ec(32)
ecdhe_pk ← *xP*

- No caching
- No additional input

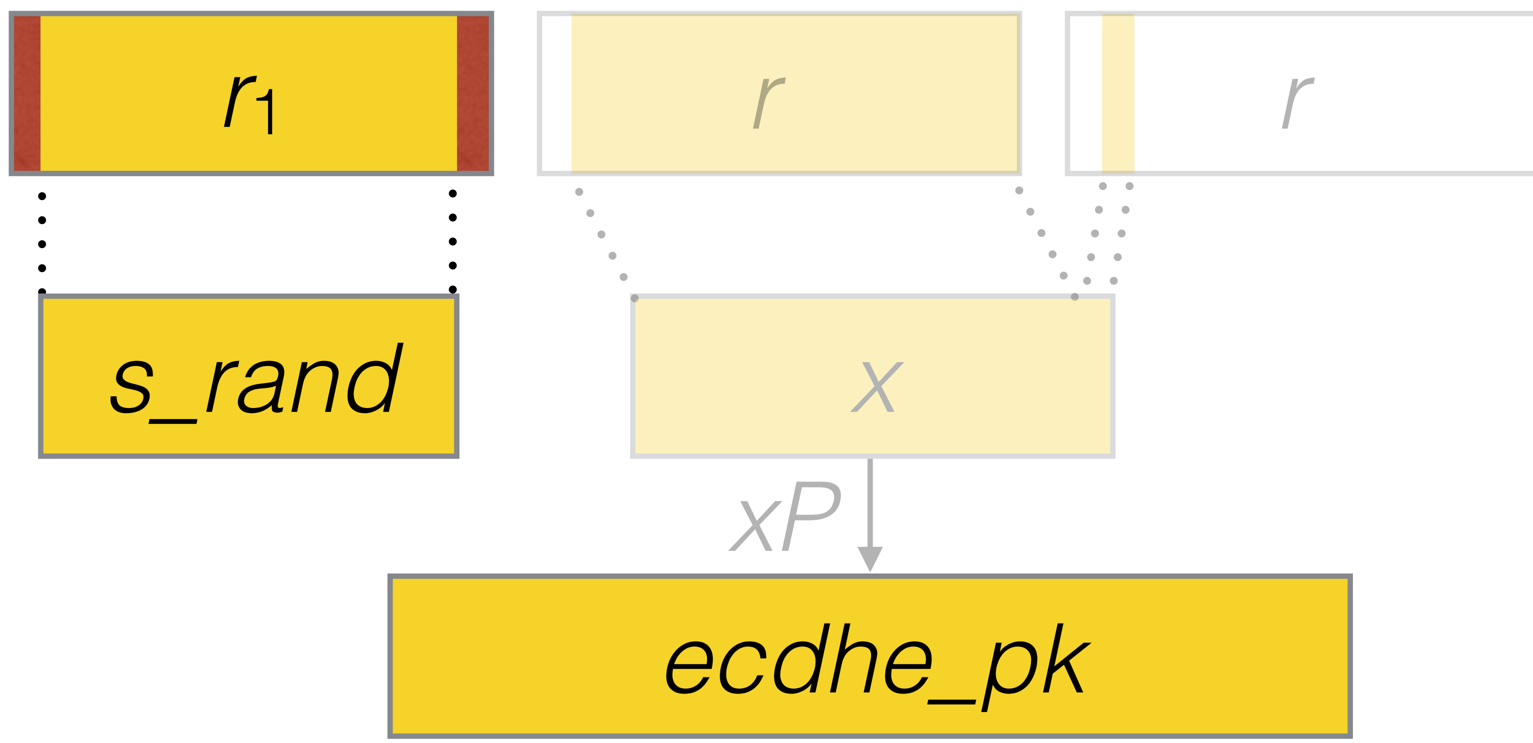
RSA BSAFE Share for Java



server_rand ← dual_ec(28)
x ← dual_ec(32)
ecdhe_pk ← *xP*

- No caching
- No additional input

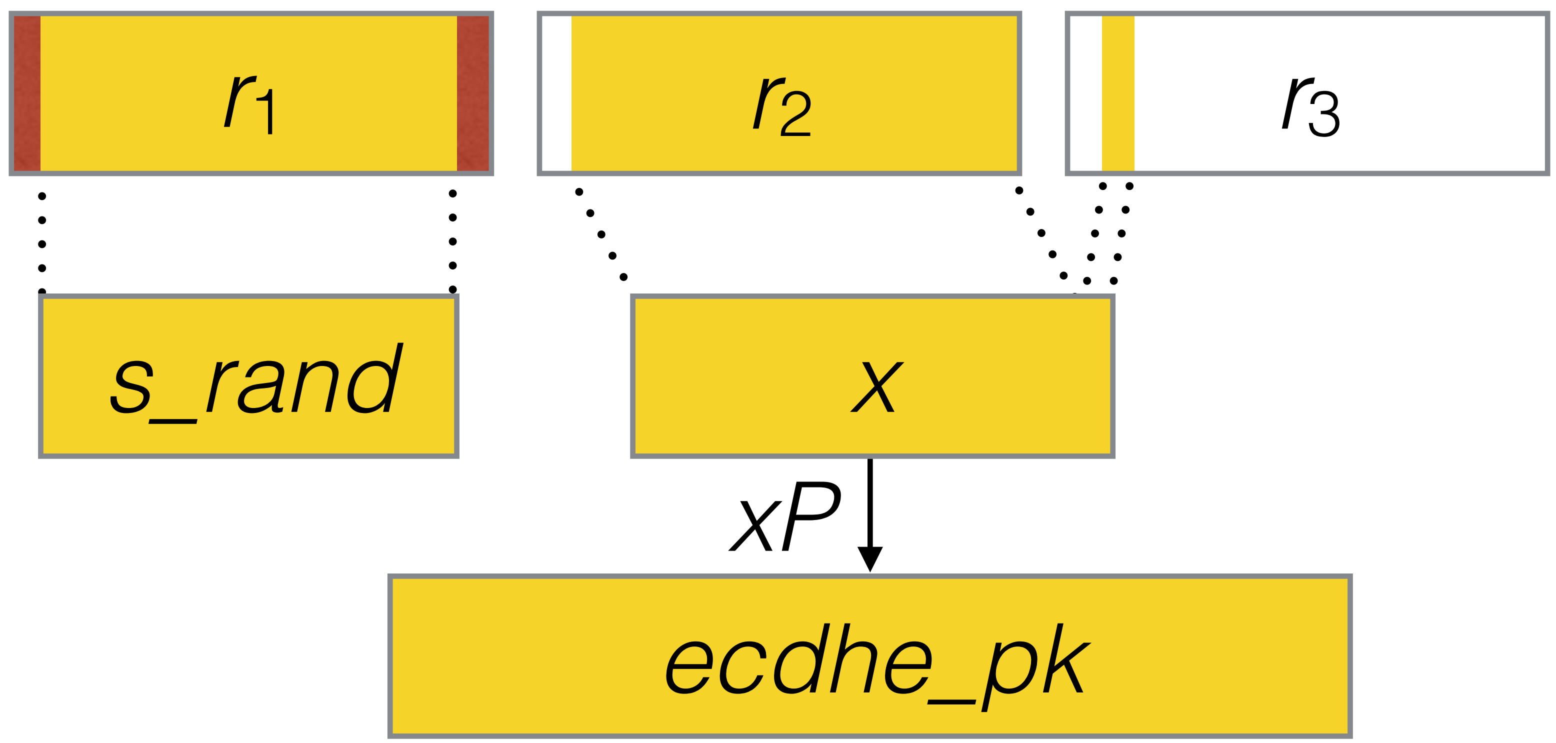
RSA BSAFE Share for Java



server_rand ← dual_ec(28)
x ← dual_ec(32)
ecdhe_pk ← *xP*

- No caching
- No additional input

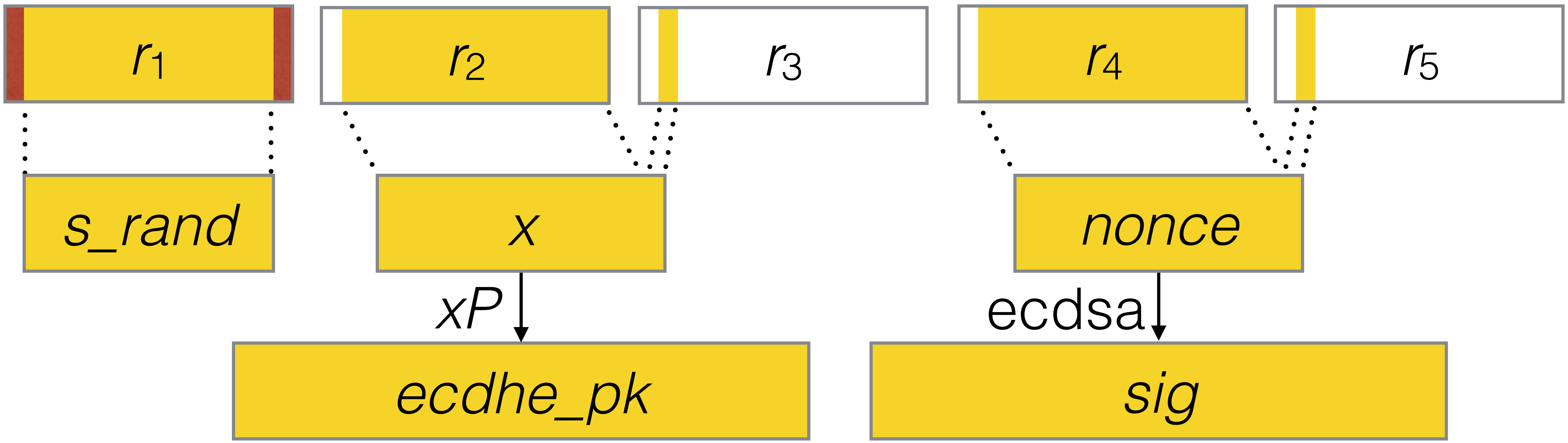
RSA BSAFE Share for Java



server_rand ← dual_ec(28)
x ← dual_ec(32)
ecdhe_pk ← *xP*

- No caching
- No additional input

RSA BSAFE Share for Java



server_rand ← dual_ec(28)
x ← dual_ec(32)
ecdhe_pk ← *xP*
nonce ← dual_ec(32)
sig ← ecdsa(*key*, *nonce*, *params*)

- Recovering *nonce* allows computing the long-term signing key

Attack validation

Attack validation

- Generate new point *Q*’

Attack validation

- Generate new point *Q*’
- Replace *Q* with *Q*’ (incl. tables of multiples of *Q*)

Attack validation

- Generate new point *Q*'
- Replace *Q* with *Q*' (incl. tables of multiples of *Q*)
- R.E. BSAFE Java, BSAFE C/C++, SChannel

Attack validation

- Generate new point *Q*'
- Replace *Q* with *Q*' (incl. tables of multiples of *Q*)
- R.E. BSAFE Java, BSAFE C/C++, SChannel
- Capture network traces with **`tcpdump`**

Attack validation

- Generate new point *Q*'
- Replace *Q* with *Q*' (incl. tables of multiples of *Q*)
- R.E. BSAFE Java, BSAFE C/C++, SChannel
- Capture network traces with **`tcpdump`**
- Recover TLS master secret and decrypt

Implementation choices

- TLS choices:
  - Order and size of server random, session id, and (EC)DHE private key generation
  - Session id random or not

- Dual EC choices:
  - Caching unused generated bytes
  - Additional input hashed into PRNG state
  - Dual EC 2006 or Dual EC 2007

	Generation order	Size	Caching	Additional input	Version
RSA Java	server random ecdhe sk ecdsa nonce	28 32 32	No	No	2007
RSA C/C++	s. rand || session id dhe sk dsa nonce	60 20 20	Yes	No	2007
Microsoft SChannel	session id ecdhe sk other server random ecdsa nonce	32 40 32 28 32	No	No	2006*
OpenSSL-fixed	session id server random ecdhe sk ecdsa nonce	32 28 32 32	No	Yes: sec || μs || ctr || pid	2007

* Due to a bug

Attack summary

	Default PRNG	Bytes per session	Additional input entropy (bits)	Time* (min)
RSA Java	✔	28	—	63.96
RSA C/C++	✔	31–60	—	0.04
Microsoft SChannel I		28	—	62.97
Microsoft SChannel II		30	—	182.64
OpenSSL-fixed I		32	20	0.02
OpenSSL-fixed III		32	35 + k	2

* 4 node cluster

All the pieces matter

- Exploitability of a PRNG depends on
  - PRNG design
  - Protocol design
  - Implementation choices

All the pieces matter

- Exploitability of a PRNG depends on
  - PRNG design
  - Protocol design
  - Implementation choices
- It helps to have a hand in all three
  - NSA designed Dual EC
  - NSA wrote TLS extensions which facilitate attack
  - NSA paid RSA $10M to make Dual EC the default

Demo

"I also think that the mathematics behind the papers on breaking [Dual EC] are not very realistic."

– Richard "Dickie" George

Former Technical Director of the NSA Information Assurance Directorate

Thank you!

http://dualec.org