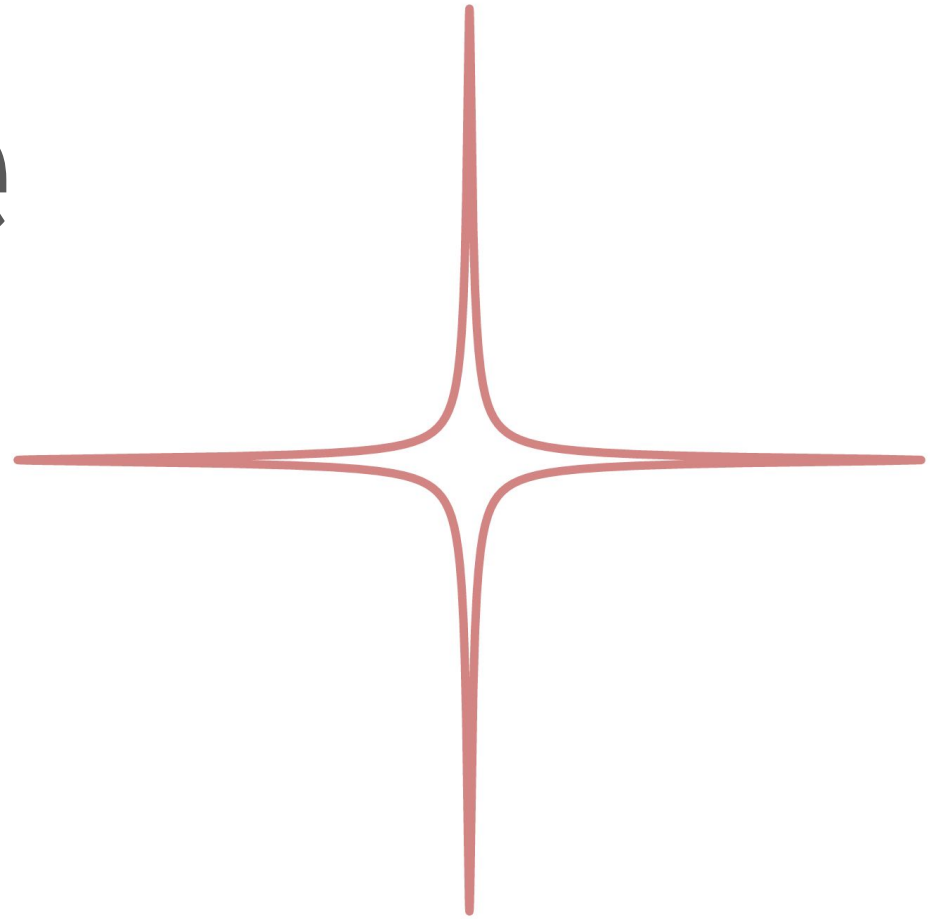
State of the Curve

DEFCON 2016



Deirdre Connolly
@durumcrustulum

Backdoored RNG
New IETF Standards
Post-Quantum Crypto

Backdoored RNG
New IETF Standards
Post-Quantum Crypto

2015-12 Out of Cycle Security Bulletin: ScreenOS: Multiple Security issues with ScreenOS (CVE-2015-7755, CVE-2015-7756)

[JSA10713] Show KB Properties

PRODUCT AFFECTED:

Please see below for details.

PROBLEM:

During an internal code review, two security issues were identified.

Administrative Access (CVE-2015-7755) allows unauthorized remote administrative access to the device. Exploitation of

Example:

Normal login by user **username1**:

```
2015-12-17 09:00:00 system warn 00515 Admin user username1 has logged on via SSH from …..
2015-12-17 09:00:00 system warn 00528 SSH: Password authentication successful for admin
user 'username1' at host …
```

Compromised login by user **username2**:

```
2015-12-17 09:00:00 system warn 00515 Admin user system has logged on via SSH from …..
2015-12-17 09:00:00 system warn 00528 SSH: Password authentication successful for admin
user 'username2' at host …
```

Note that a skilled attacker would likely remove these entries from the local log file, thus effectively eliminating any reliable signature that the device had been compromised.

This issue has been assigned CVE-2015-7755.

VPN Decryption (CVE-2015-7756) may allow a knowledgeable attacker who can monitor VPN traffic to decrypt that traffic. It is independent of the first issue.

This issue affects ScreenOS 6.2.0r15 through 6.2.0r18 and 6.3.0r12 through 6.3.0r20. **No other Juniper products or versions of ScreenOS are affected by this issue.**

There is no way to detect that this vulnerability was exploited.

This issue has been assigned CVE-2015-7756.

Juniper SIRT is not aware of any malicious exploitation of these vulnerabilities, however the password needed for the administrative access has been revealed publicly.

No other Juniper Networks products or platforms are affected by these issues.

Juniper has issued a statement about these vulnerabilities at: http://forums.juniper.net/t5/Security-Incident-Response/bg-p/SIRT

SOLUTION:

Example:

Normal login by user **username1**:

```
2015-12-17 09:00:00 system warn 00515 Admin user username1 has logged on via SSH from …..
2015-12-17 09:00:00 system warn 00528 SSH: Password authentication successful for admin
user 'username1' at host …
```

Compromised login by user **username2**:

```
2015-12-17 09:00:00 system warn 00515 Admin user system has logged on via SSH from …..
2015-12-17 09:00:00 system warn 00528 SSH: Password authentication successful for admin
user 'username2' at host …
```

Note that a skilled attacker would likely remove these entries from the local log file, thus effectively eliminating any reliable signature that the device had been compromised.

This issue has been assigned CVE-2015-7755.

VPN Decryption (CVE-2015-7756) may allow a knowledgeable attacker who can monitor VPN traffic to decrypt that traffic. It is independent of the first issue.

This issue affects ScreenOS 6.2.0r15 through 6.2.0r18 and 6.3.0r12 through 6.3.0r20. **No other Juniper products or versions of ScreenOS are affected by this issue.**

There is no way to detect that this vulnerability was exploited.

This issue has been assigned CVE-2015-7756.

Juniper SIRT is not aware of any malicious exploitation of these vulnerabilities, however the password needed for the administrative access has been revealed publicly.

No other Juniper Networks products or platforms are affected by these issues.

Juniper has issued a statement about these vulnerabilities at: http://forums.juniper.net/t5/Security-Incident-Response/bg-p/SIRT

SOLUTION:

BOMBSHELL! —

“Unauthorized code” in Juniper firewalls decrypts encrypted VPN traffic

Backdoor in NetScreen firewalls gives attackers admin access, VPN decrypt ability.

DAN GOODIN - 12/17/2015, 3:50 PM

133

An operating system used to manage firewalls sold by Juniper Networks contains unauthorized code that surreptitiously decrypts traffic sent through virtual private networks, officials from the company warned Thursday.

It's not clear how the code got there or how long it has been there. An advisory published by the company said that NetScreen firewalls using ScreenOS 6.2.0r15 through 6.2.0r18 and 6.3.0r12 through 6.3.0r20 are affected and require immediate patching. Release notes published by Juniper suggest the earliest vulnerable versions date back to at least 2012 and possibly earlier. There's no evidence right now that the backdoor was put in other Juniper OSes or devices.

"During a recent internal code review, Juniper discovered unauthorized code in ScreenOS that

TECH CYBERSECURITY REPORT

Who hid a backdoor in a popular corporate VPN program?

By Ashley Carman on December 22, 2015 08:33 am Email @ashleyrcarman

THE LATEST HEADLINES

Apple celebrates diversity in new iPhone ad to air during the Olympics

Your TV picture may be clear, but settings are opaque

KIM ZETTER SECURITY 12.18.15 6:35 PM

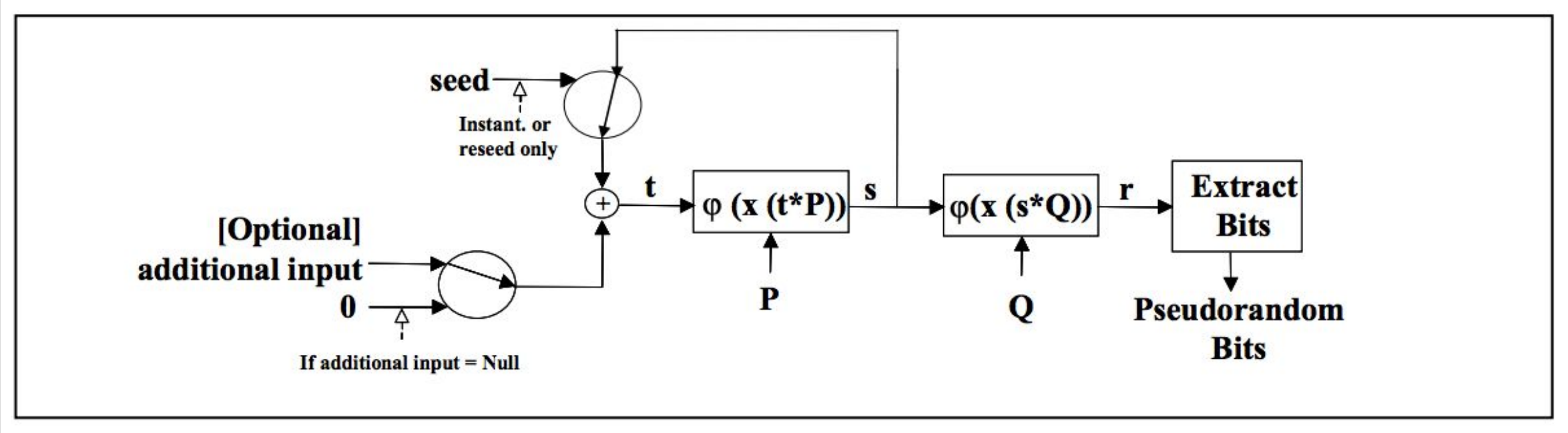
SECRET CODE FOUND IN JUNIPER'S FIREWALLS SHOWS RISK OF GOVERNMENT BACKDOORS

patched

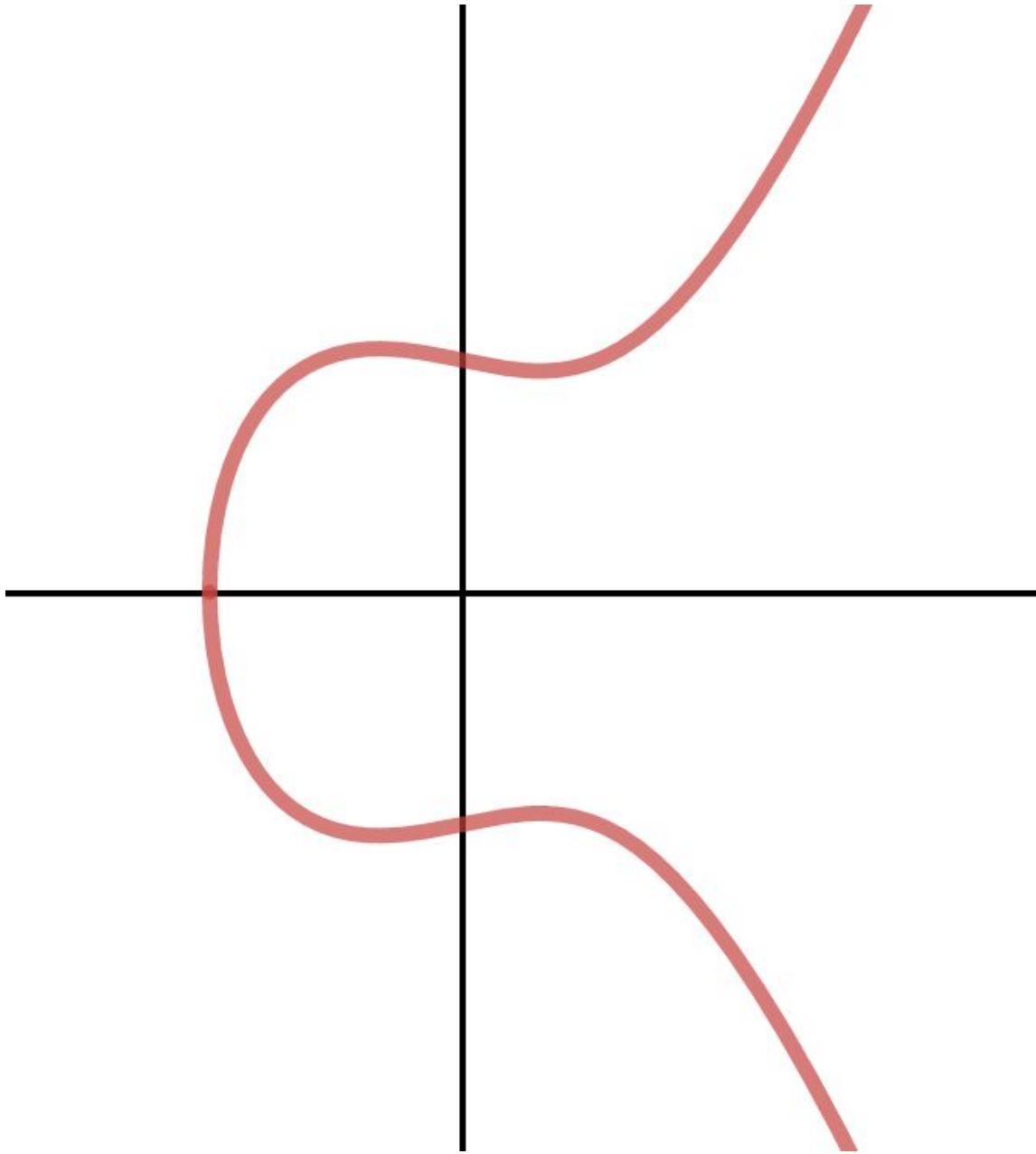
vulnerable

Credit: David Wong, cryptologie.net

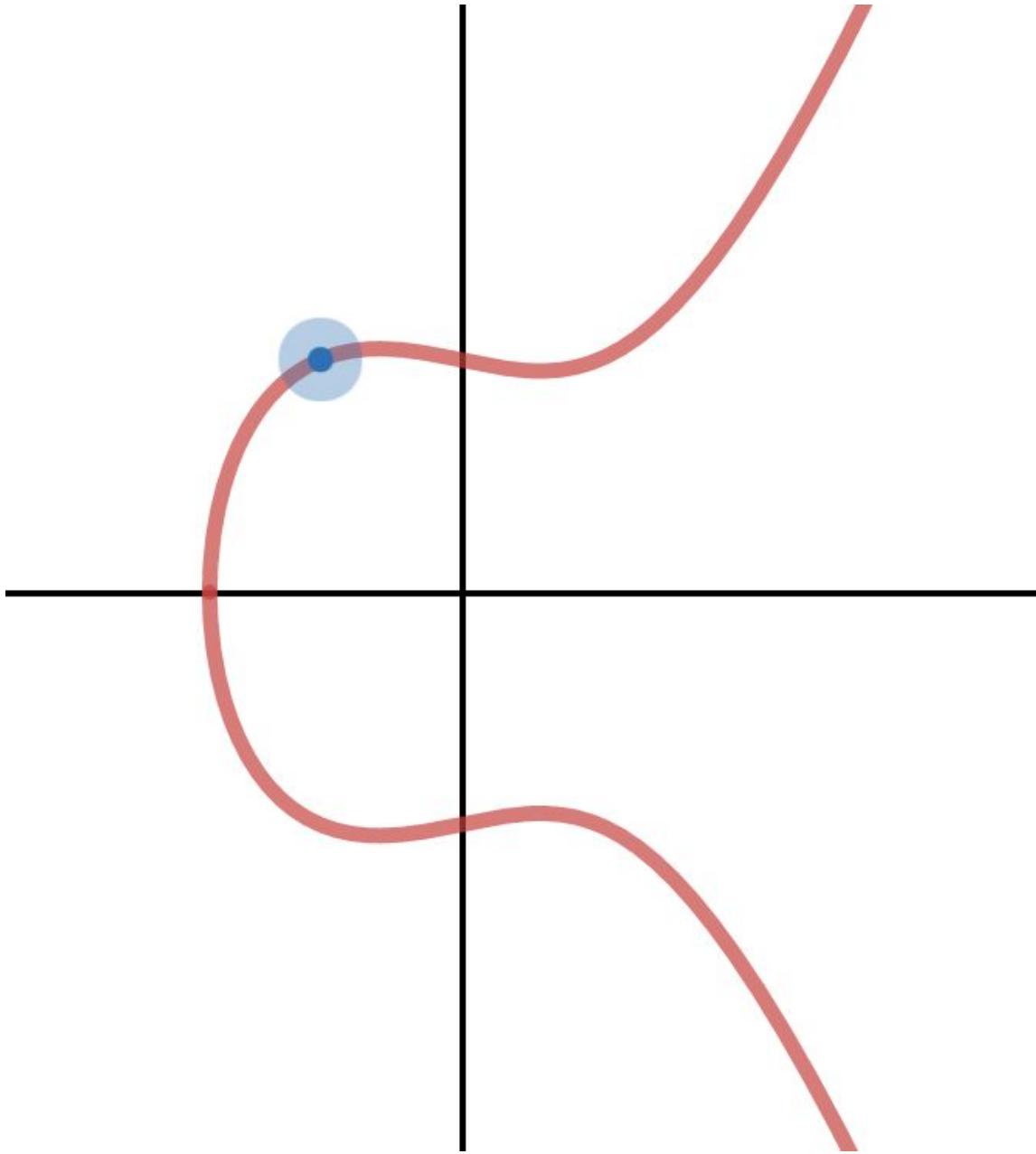
Dual_EC_DRBG



http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf

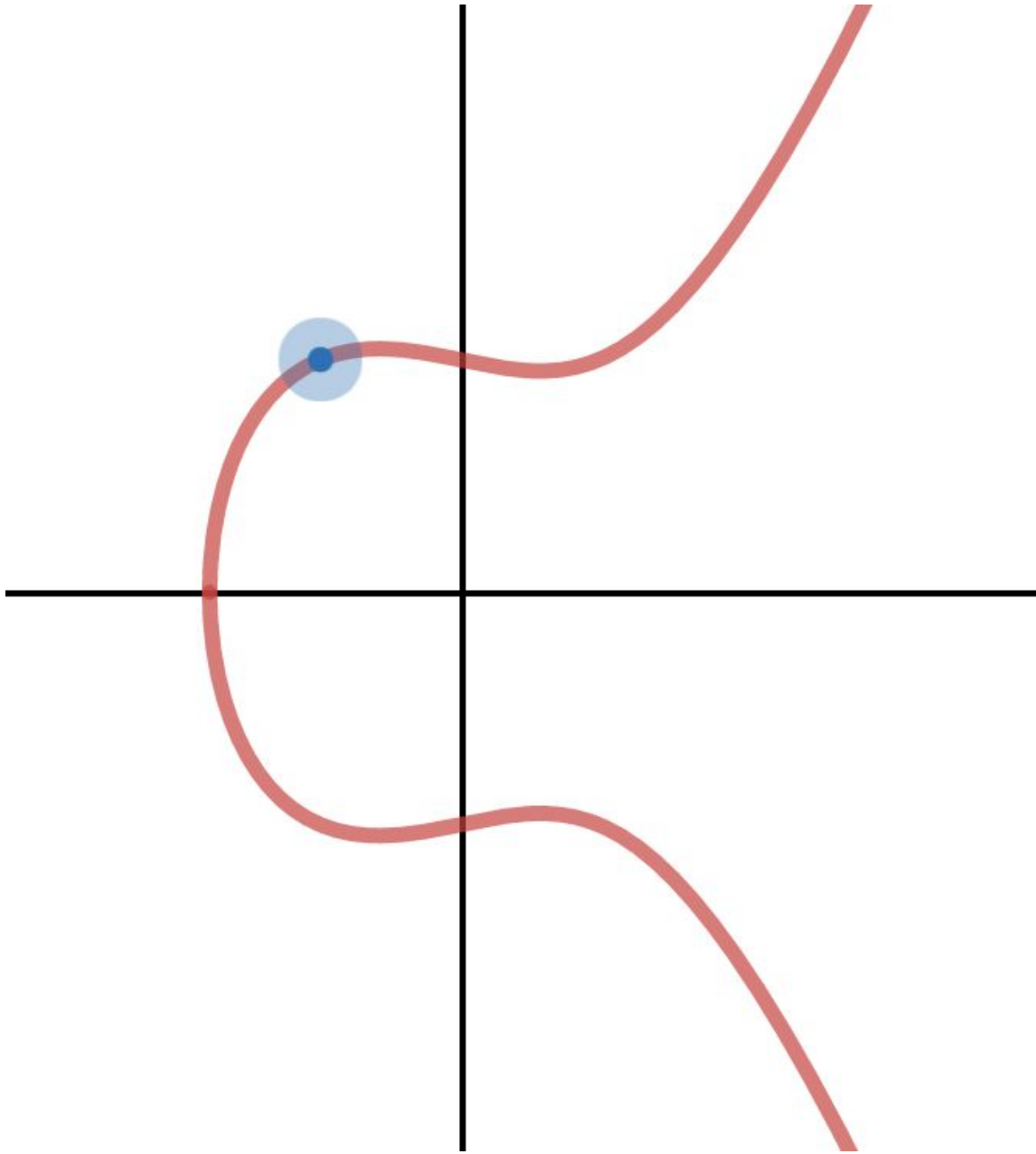


$y^2=x^3-ax+b$



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

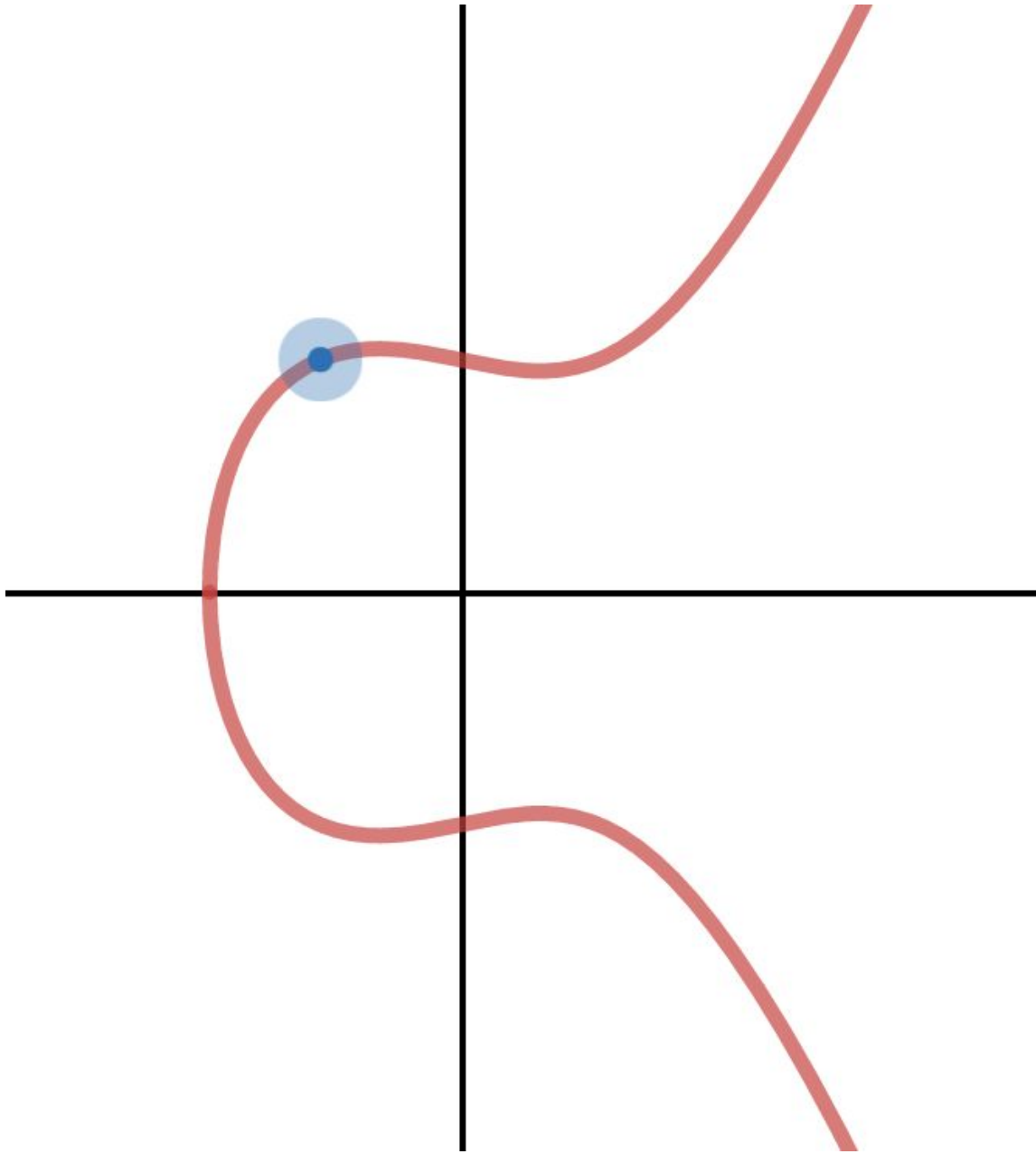
$$P + \mathcal{O} = P$$



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

$$P + \mathcal{O} = P$$

$$P + -P = \mathcal{O}$$

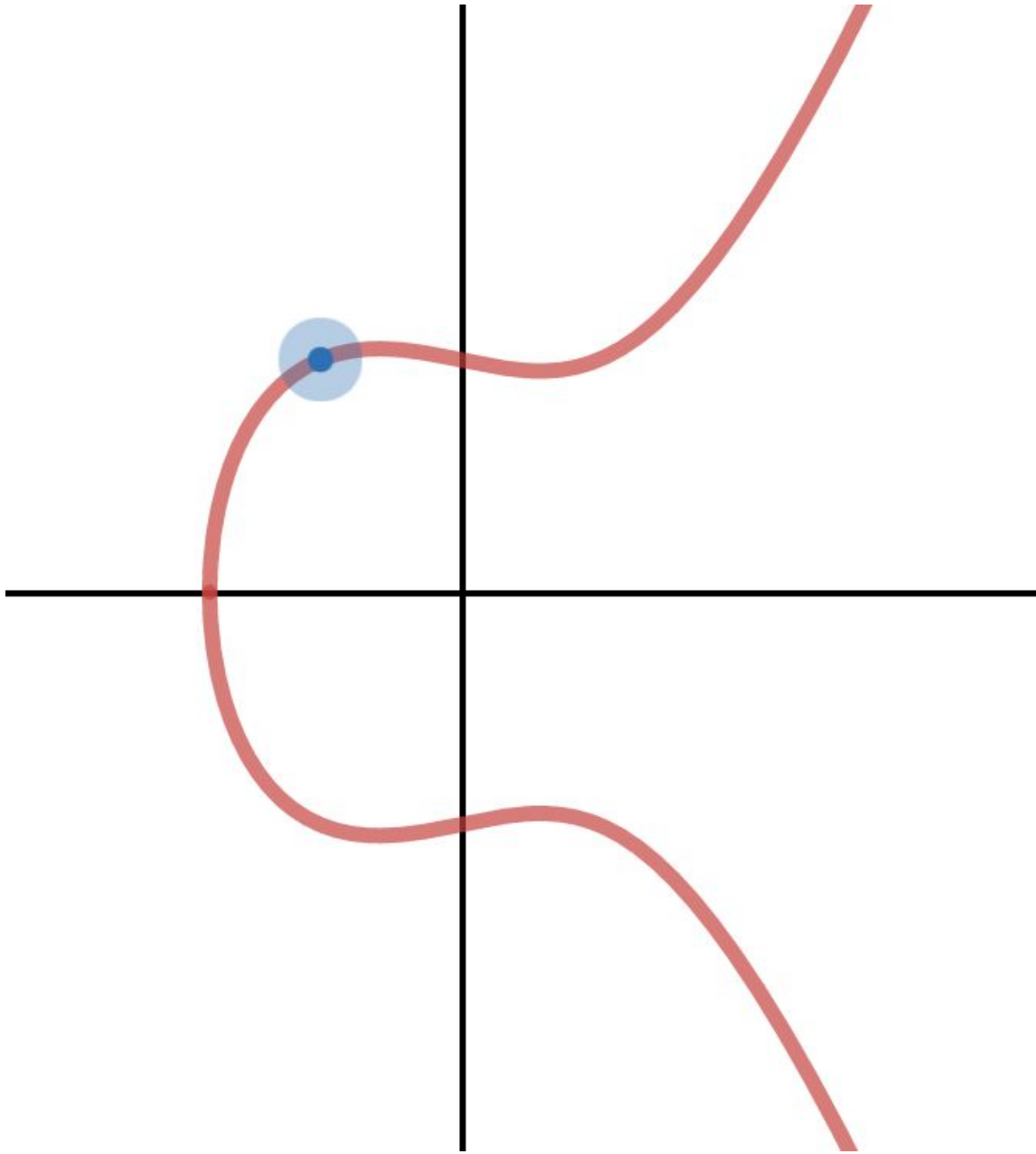


$y^2=x^3-ax+b$

$$P + \mathcal{O} = P$$

$$P + -P = \mathcal{O}$$

$$P + Q + R = \mathcal{O}$$



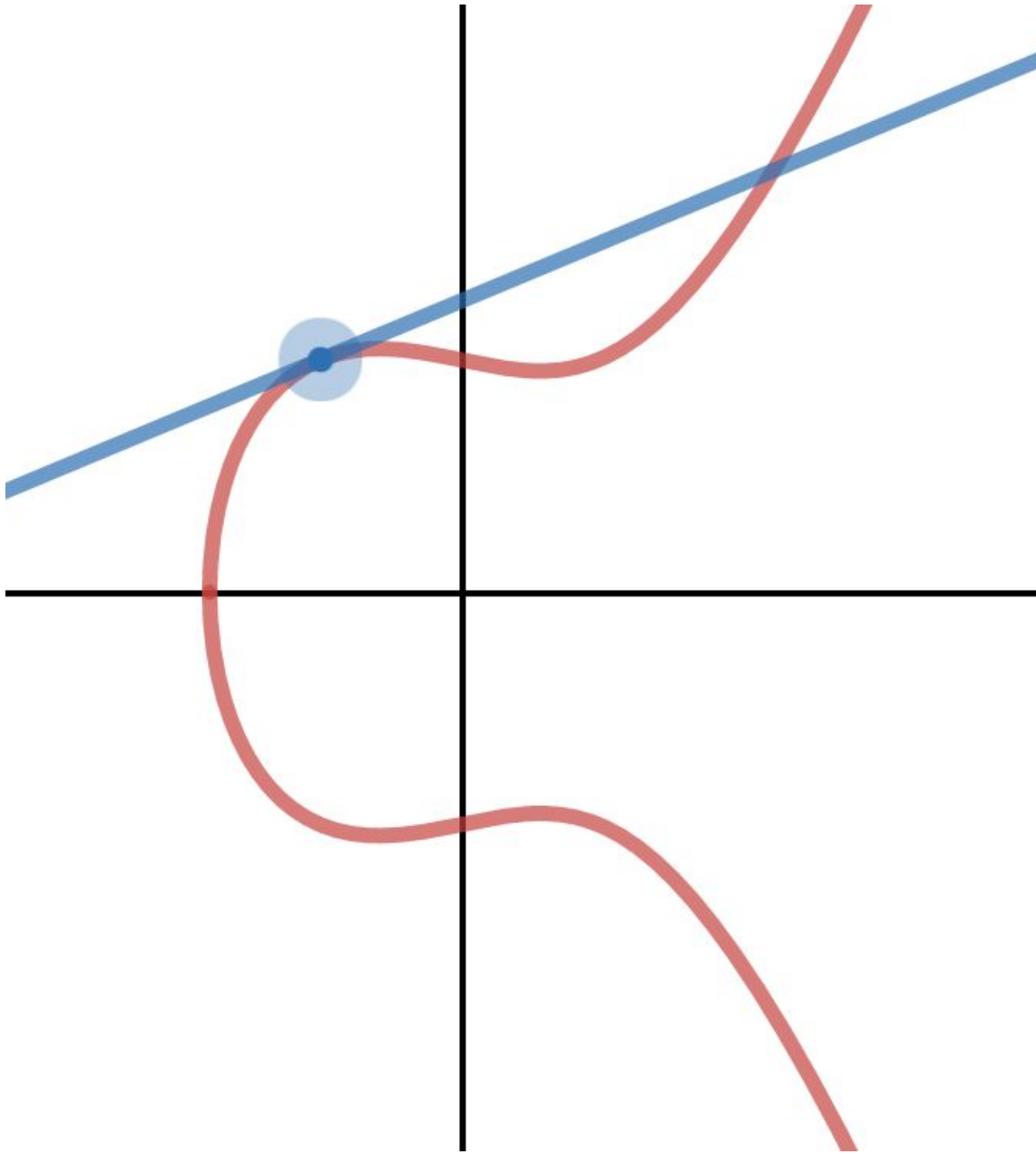
$y^2=x^3-ax+b$

$$P + \mathcal{O} = P$$

$$P + -P = \mathcal{O}$$

$$P + Q + R = \mathcal{O}$$

$$P + P + R = \mathcal{O}$$



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

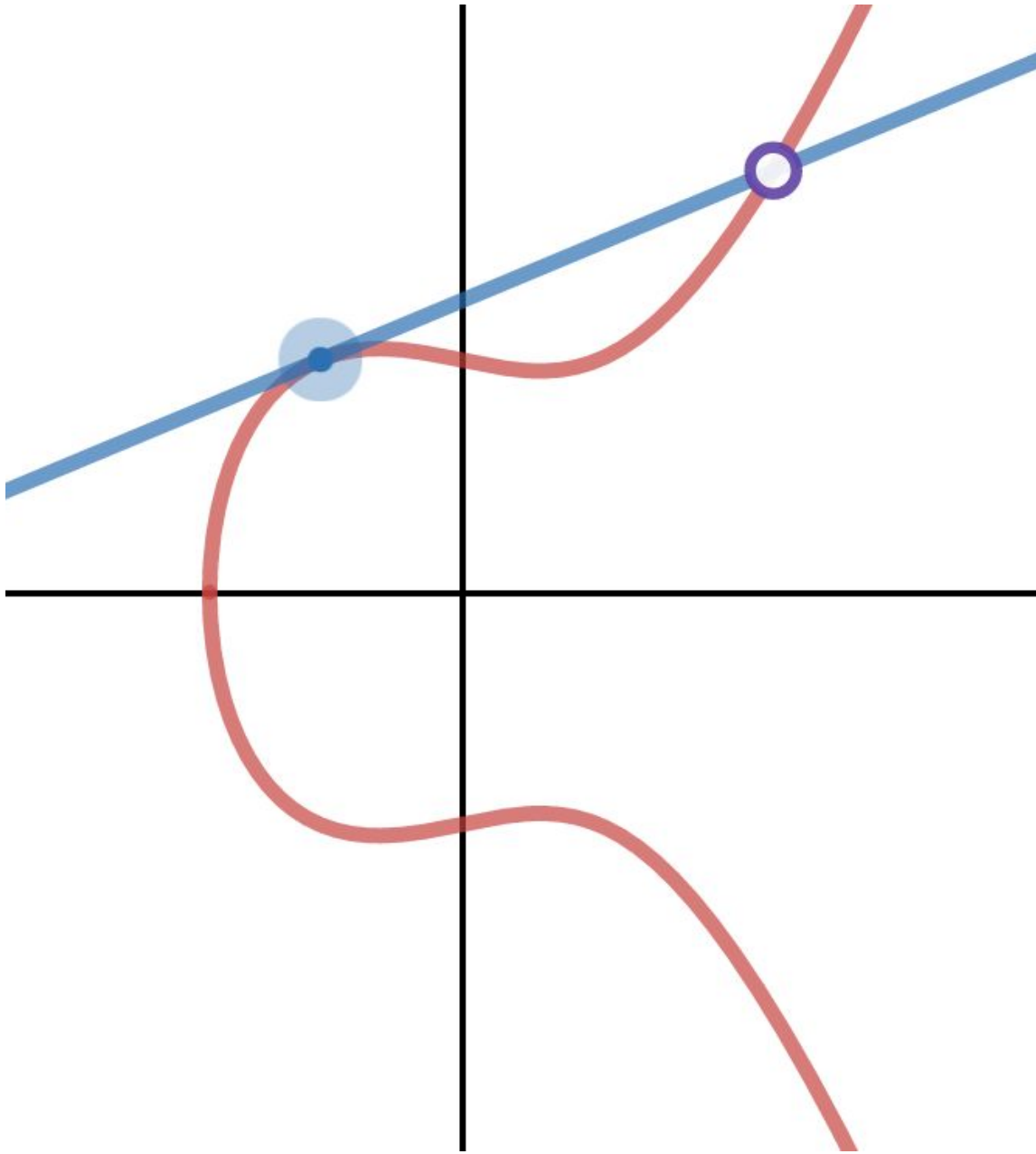
$$P + \mathcal{O} = P$$

$$P + -P = \mathcal{O}$$

$$P + Q + R = \mathcal{O}$$

$$P + P + R = \mathcal{O}$$

$$P + P = -R = 2P$$



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

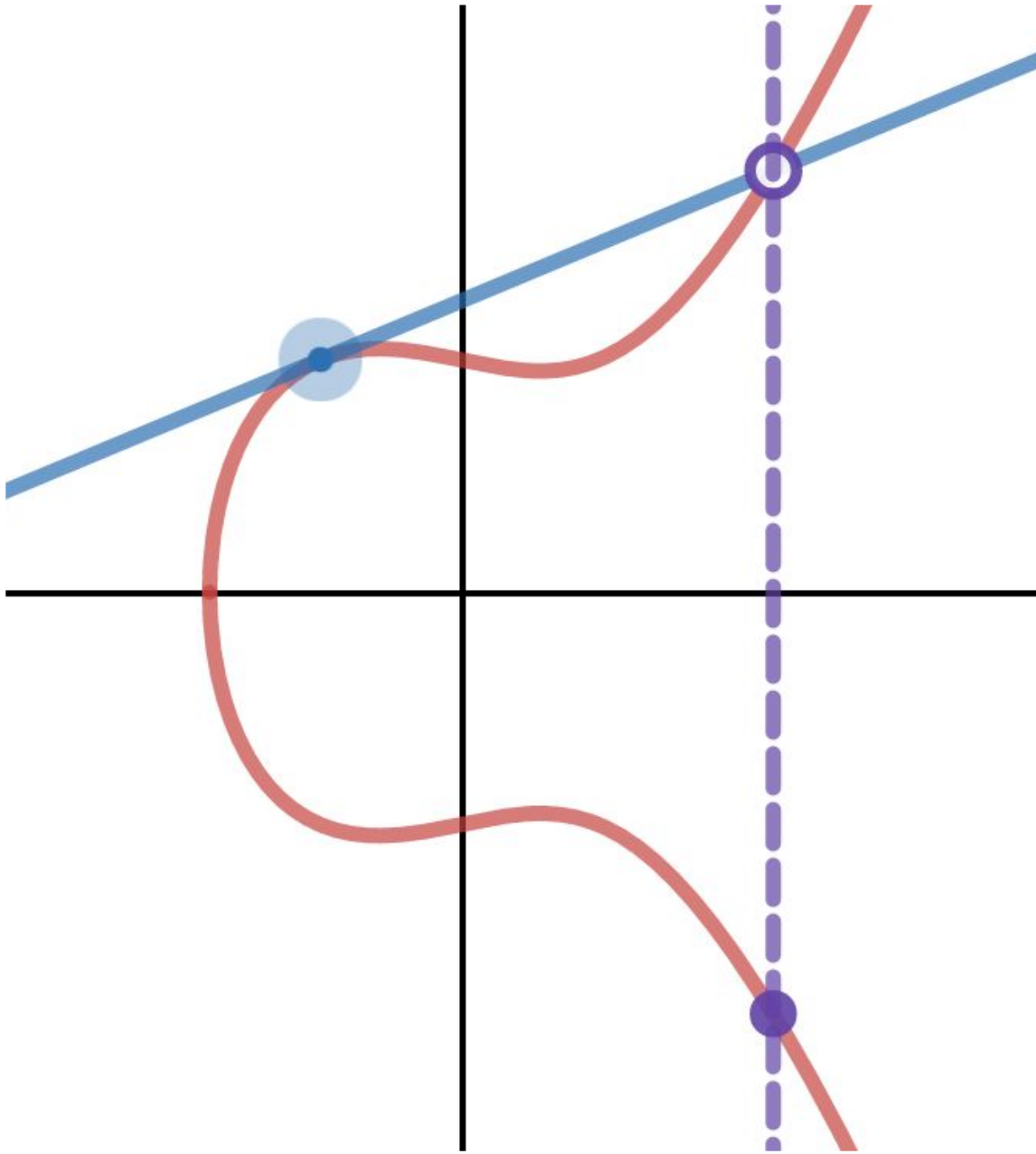
$$P + \mathcal{O} = P$$

$$P + -P = \mathcal{O}$$

$$P + Q + R = \mathcal{O}$$

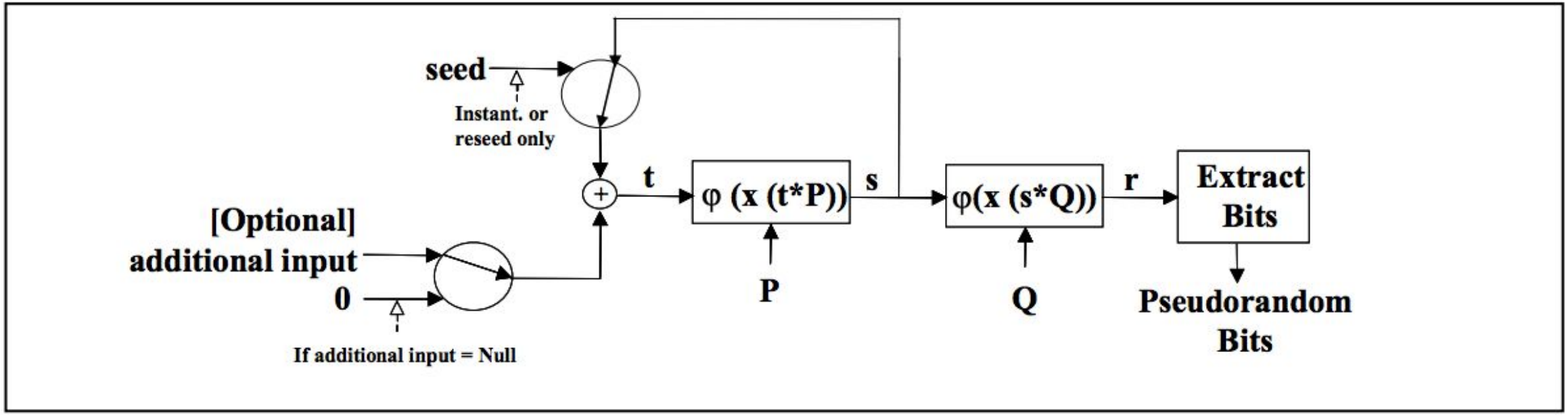
$$P + P + R = \mathcal{O}$$

$$P + P = -R = 2P$$



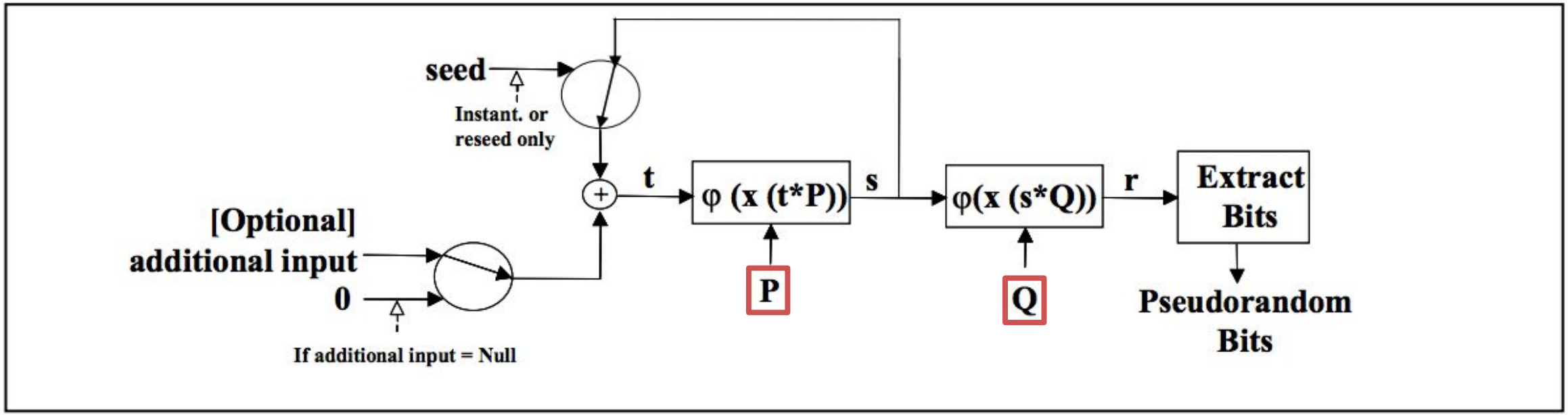
$y^2=x^3-ax+b$

Dual_EC_DRBG



http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf

Dual_EC_DRBG



http://csrc.nist.gov/publications/nistpubs/800-90A/SP800-90A.pdf

On the Possibility of a Back Door in the NIST SP800-90 Dual Ec Prng

Dan Shumow

Niels Ferguson

Microsoft

CRYPTO 2007

$$If\ lQ = \mathcal{O},$$

$$If\ lQ = \mathcal{O},\ ord(Q) = l$$

$$If\ lQ = \mathcal{O},\ ord(Q) = l$$

$$Q = eP \qquad d = e^{-1}\ mod\ l$$

$$If\ lQ = \mathcal{O},\ ord(Q) = l$$

$$Q = eP \qquad d = e^{-1}\ mod\ l$$

$$dQ$$

$$If\ lQ = \mathcal{O},\ ord(Q) = l$$

$$Q = eP \qquad d = e^{-1}\ mod\ l$$

$$dQ = e^{-1}eP\ mod\ l$$

$$If\ lQ = \mathcal{O},\ ord(Q) = l$$

$$Q = eP \qquad d = e^{-1}\ mod\ l$$

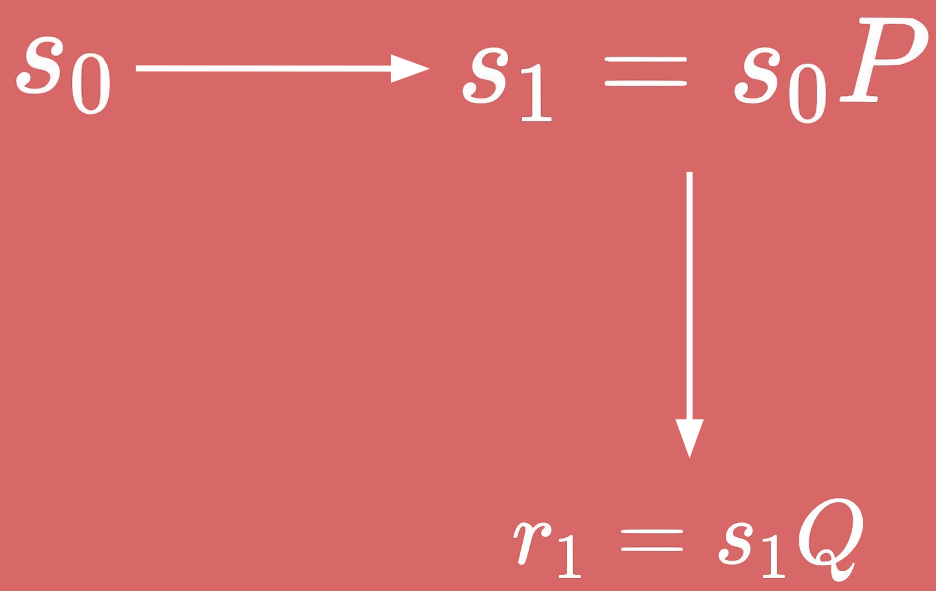
$$dQ = e^{-1}eP\ mod\ l = \mathbf{P}$$

s_0

Internal state

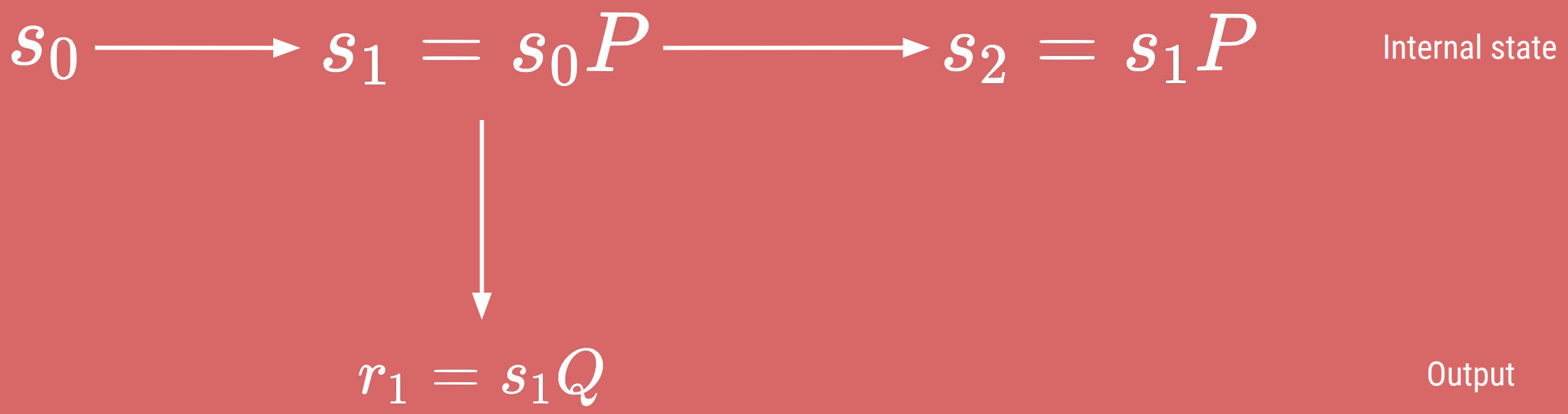
$$s_0 \longrightarrow s_1 = s_0 P$$

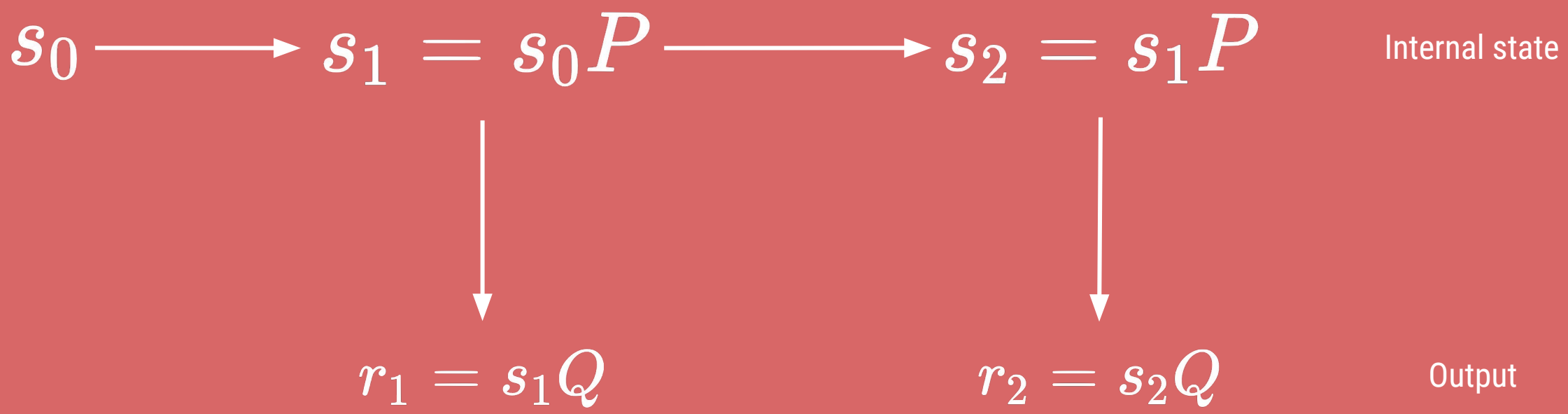
Internal state

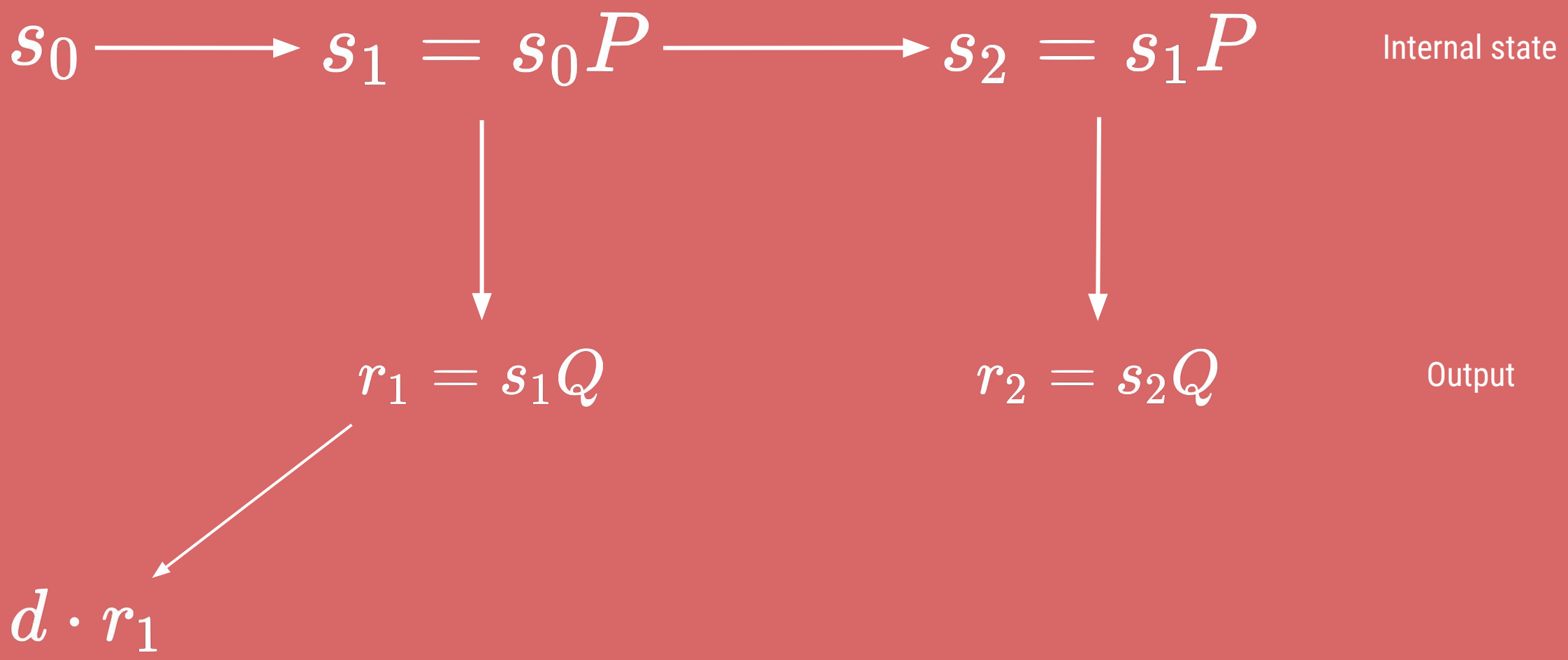


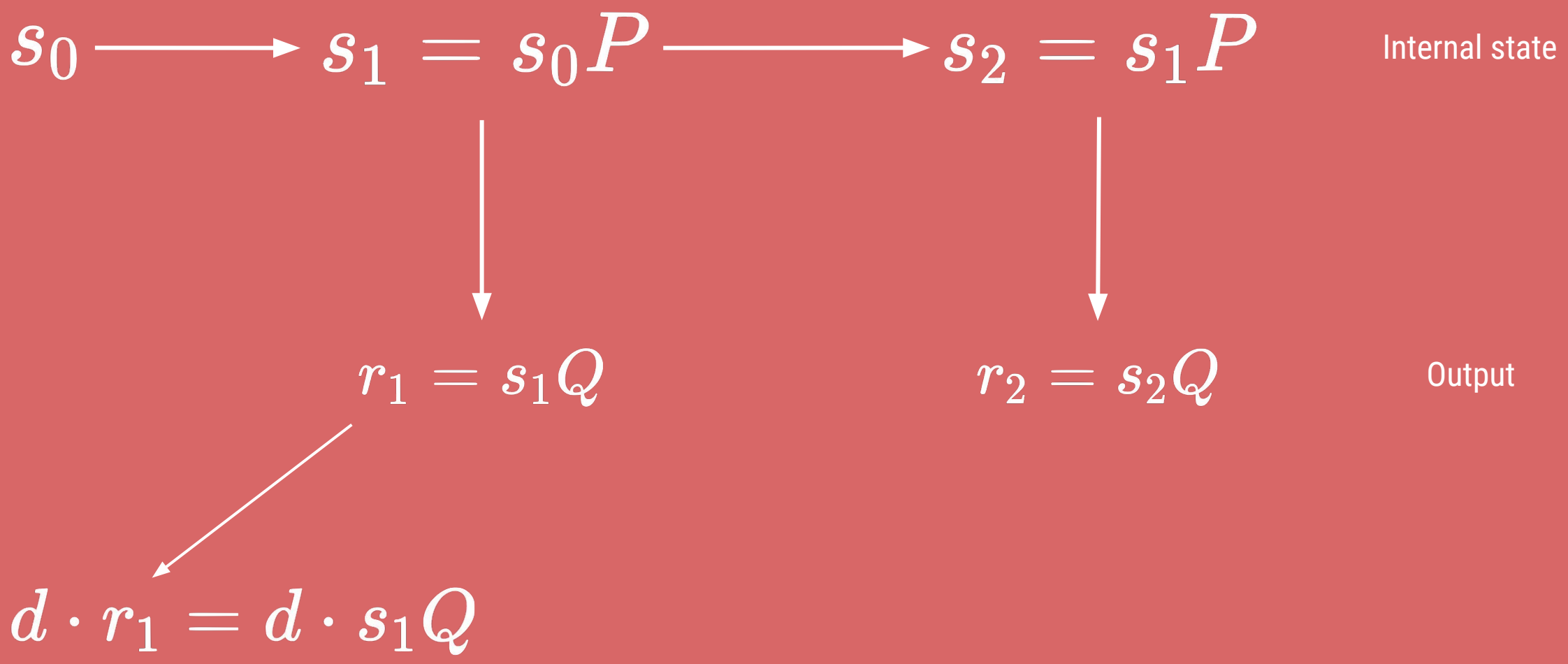
Internal state

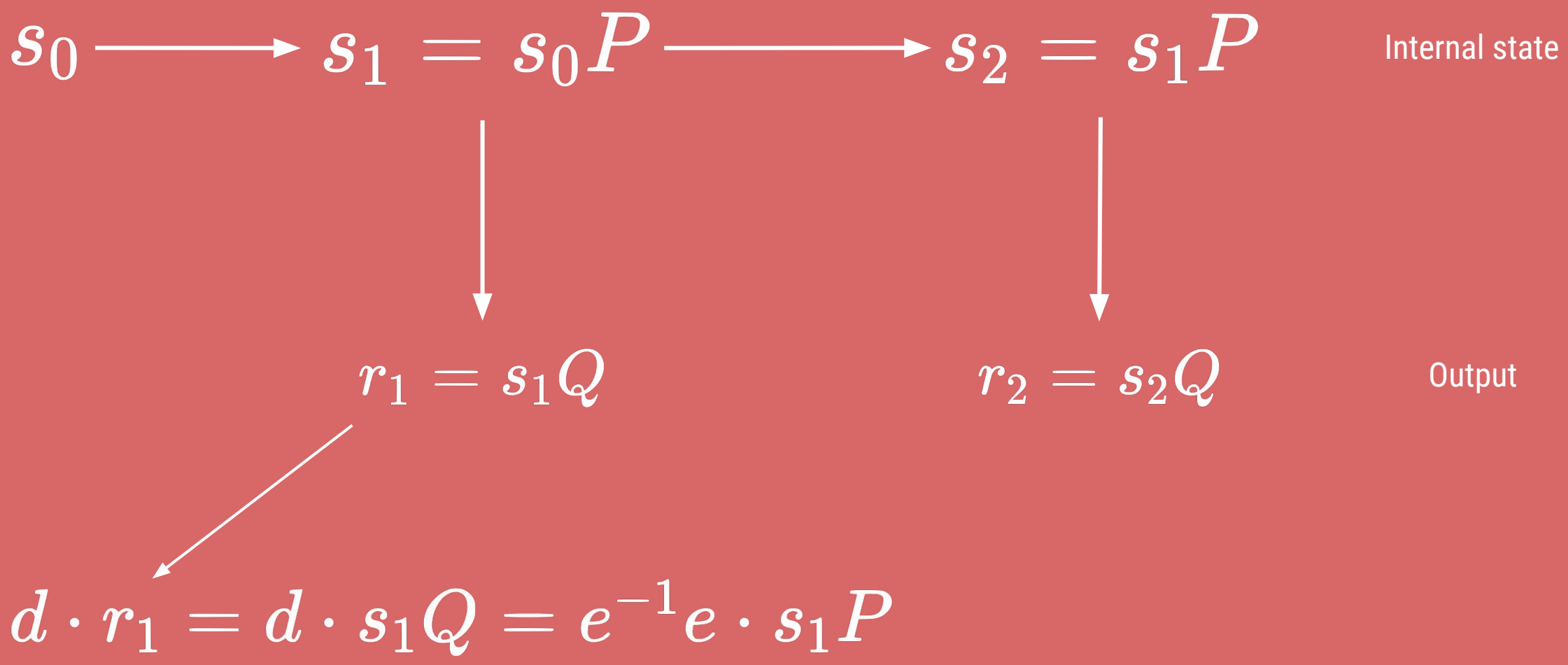
Output

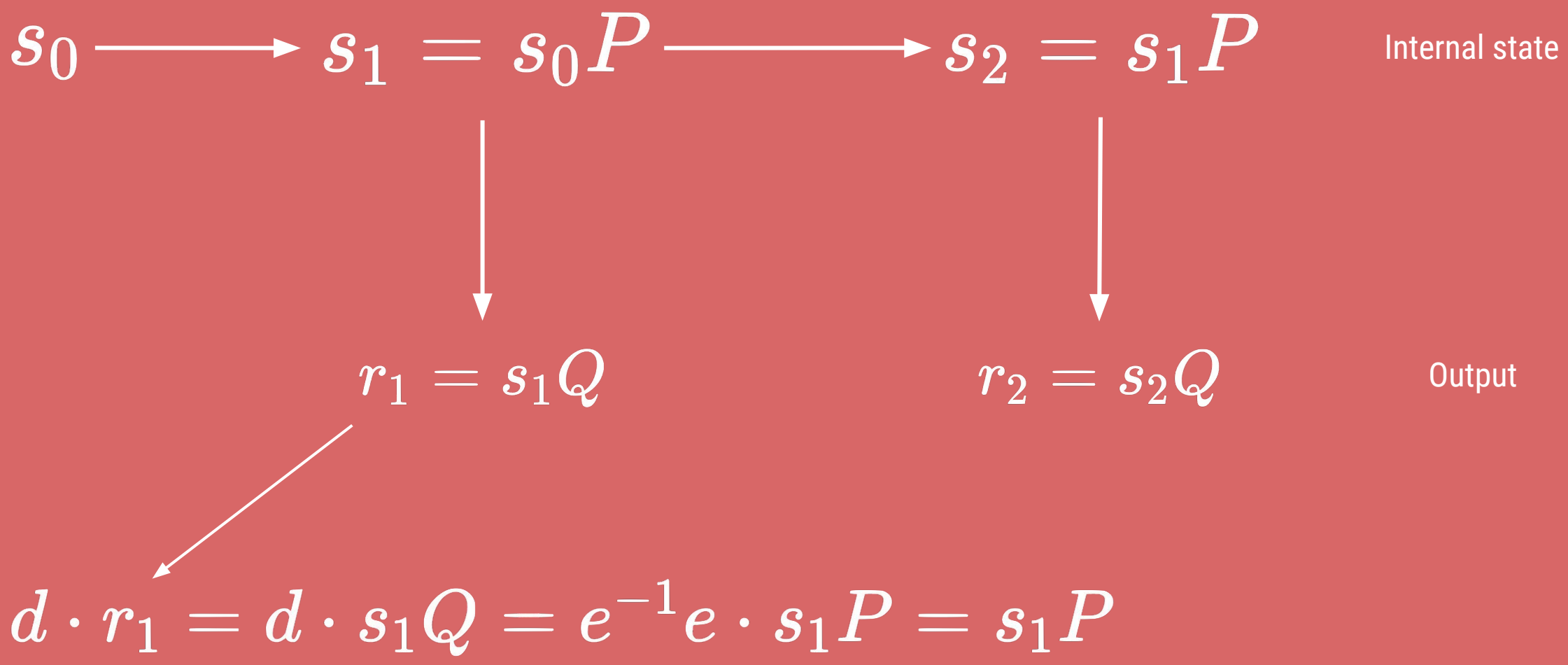

$s_0 \longrightarrow s_1 = s_0P \longrightarrow s_2 = s_1P$
Internal state
$r_1 = s_1Q$
Output

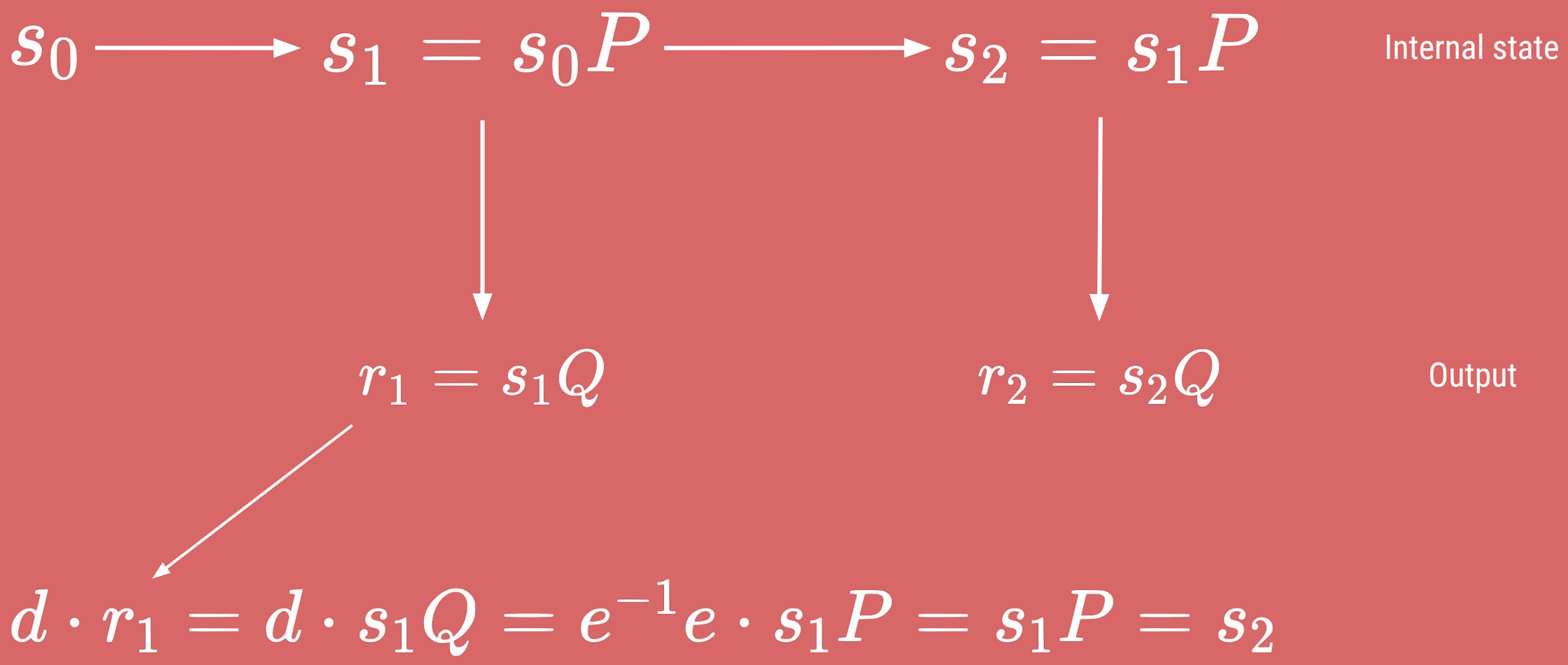

s_0
$s_1 = s_0 P$
$s_2 = s_1 P$
Internal state
$r_1 = s_1 Q$
$r_2 = s_2 Q$
Output

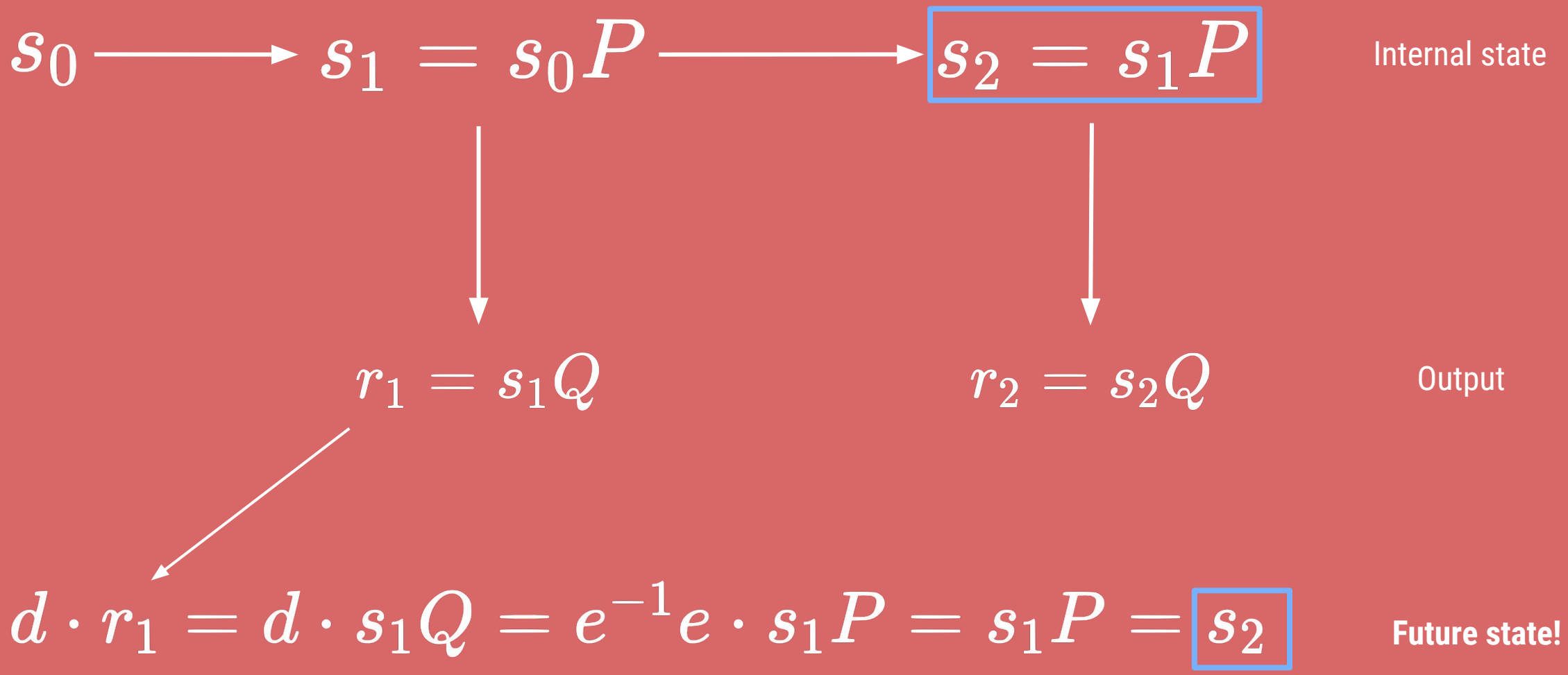

s_0
$s_1 = s_0 P$
$s_2 = s_1 P$
Internal state
$r_1 = s_1 Q$
$r_2 = s_2 Q$
Output
$d \cdot r_1$


s_0
$s_1 = s_0 P$
$s_2 = s_1 P$
Internal state
$r_1 = s_1 Q$
$r_2 = s_2 Q$
Output
$d \cdot r_1 = d \cdot s_1 Q$


s_0
$s_1 = s_0 P$
$s_2 = s_1 P$
Internal state
$r_1 = s_1 Q$
$r_2 = s_2 Q$
Output
$d \cdot r_1 = d \cdot s_1 Q = e^{-1} e \cdot s_1 P$


s_0
$s_1 = s_0 P$
$s_2 = s_1 P$
Internal state
$r_1 = s_1 Q$
$r_2 = s_2 Q$
Output
$d \cdot r_1 = d \cdot s_1 Q = e^{-1} e \cdot s_1 P = s_1 P$


s_0
$s_1 = s_0 P$
$s_2 = s_1 P$
Internal state
$r_1 = s_1 Q$
$r_2 = s_2 Q$
Output
$d \cdot r_1 = d \cdot s_1 Q = e^{-1} e \cdot s_1 P = s_1 P = s_2$


$s_0 \longrightarrow s_1 = s_0 P \longrightarrow \boxed{s_2 = s_1 P}$
Internal state
$r_1 = s_1 Q$
$r_2 = s_2 Q$
Output
$d \cdot r_1 = d \cdot s_1 Q = e^{-1} e \cdot s_1 P = s_1 P = \boxed{s_2}$
Future state!

Critics See Efforts by Counties and Towns to Purge Minority Voters From Rolls

Pacific Northwest Weighs Response to Risks Posed by Oil Trains

Immu
Hope
but N

U.S.

N.S.A. Able to Foil Basic Safeguards of Privacy on Web

By NICOLE PERLROTH, JEFF LARSON and SCOTT SHANE SEPT. 5, 2013

1466

Juniper Networks product information about Dual_EC_DRBG

[KB28205] Show KB Properties

SUMMARY:

Juniper will use this document to comment on whether Dual Elliptic Curve Deterministic Random Bit Generator (aka Dual_EC_DRBG) is used by any Juniper Networks products.

SOLUTION:

Due to recent statements (top of page #2) by the US National Institute of Standards and Technology (NIST) concerning the security of the Dual_EC_DRBG cryptographic algorithm, Juniper Networks would like to make the following statements:

The following product families do not utilize Dual_EC_DRBG:

1. Junos - Any product running Junos OS
2. Junos Pulse Secure Access Service (SSL-VPN / IVE OS)
3. Junos Pulse Access Control Service (UAC)
4. Junos Pulse
5. Junos Space
6. JunosE
7. CTP/CTPView

The following product families do utilize Dual_EC_DRBG, but do not use the pre-defined points cited by NIST:

1. ScreenOS*

* ScreenOS does make use of the Dual_EC_DRBG standard, but not in a way that should be vulnerable to the possible issues described by NIST. Instead of using the NIST recommended curve points, ScreenOS uses self-generated basis points.

```
1  void prng_reseed(void) {
2    blocks_generated_since_reseed = 0;
3    if (dualec_generate(prng_temporary, 32) != 32)
4      error_handler("FIPS ERROR: PRNG failure, unable to reseed\n", 11);
5    memcpy(prng_seed, prng_temporary, 8);
6    prng_output_index = 8;
7    memcpy(prng_key, &prng_temporary[prng_output_index], 24);
8    prng_output_index = 32;
9  }
10 void prng_generate(void) {
11   int time[2];
12
13   time[0] = 0;
14   time[1] = get_cycles();
15   prng_output_index = 0;
16   ++blocks_generated_since_reseed;
17   if (!one_stage_rng())
18     prng_reseed();
19   for (; prng_output_index <= 0x1F; prng_output_index += 8) {
20     // FIPS checks removed for clarity
21     x9_31_generate_block(time, prng_seed, prng_key, prng_block);
22     // FIPS checks removed for clarity
23     memcpy(&prng_temporary[prng_output_index], prng_block, 8);
24   }
25 }
```

https://eprint.iacr.org/2016/376.pdf

```
1  void prng_reseed(void) {
2    blocks_generated_since_reseed = 0;
3    if (dualec_generate(prng_temporary, 32) != 32)
4      error_handler("FIPS ERROR: PRNG failure, unable to reseed\n", 11);
5    memcpy(prng_seed, prng_temporary, 8);
6    prng_output_index = 8;
7    memcpy(prng_key, &prng_temporary[prng_output_index], 24);
8    prng_output_index = 32;
9  }
10 void prng_generate(void) {
11   int time[2];
12
13   time[0] = 0;
14   time[1] = get_cycles();
15   prng_output_index = 0;
16   ++blocks_generated_since_reseed;
17   if (!one_stage_rng())
18     prng_reseed();
19   for (; prng_output_index <= 0x1F; prng_output_index += 8) {
20     // FIPS checks removed for clarity
21     x9_31_generate_block(time, prng_seed, prng_key, prng_block);
22     // FIPS checks removed for clarity
23     memcpy(&prng_temporary[prng_output_index], prng_block, 8);
24   }
25 }
```

https://eprint.iacr.org/2016/376.pdf

patched

vulnerable

Credit: David Wong, cryptologie.net

Backdoored RNG

New IETF Standards

Post-Quantum Crypto

New IETF Standards

Backdoored RNG

Post-Quantum Crypto

CFRG
Internet-Draft
Intended status: Informational
Expires: April 11, 2016

A. Langley
Google
M. Hamburg
Rambus Cryptography Research
S. Turner
IECA, Inc.
October 9, 2015

Elliptic Curves for Security
draft-irtf-cfrg-curves-11

Abstract

This memo specifies two elliptic curves over prime fields that offer high practical security in cryptographic applications, including Transport Layer Security (TLS). These curves are intended to operate at the ~128-bit and ~224-bit security level, respectively, and are generated deterministically based on a list of required properties.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on April 11, 2016.

Copyright Notice

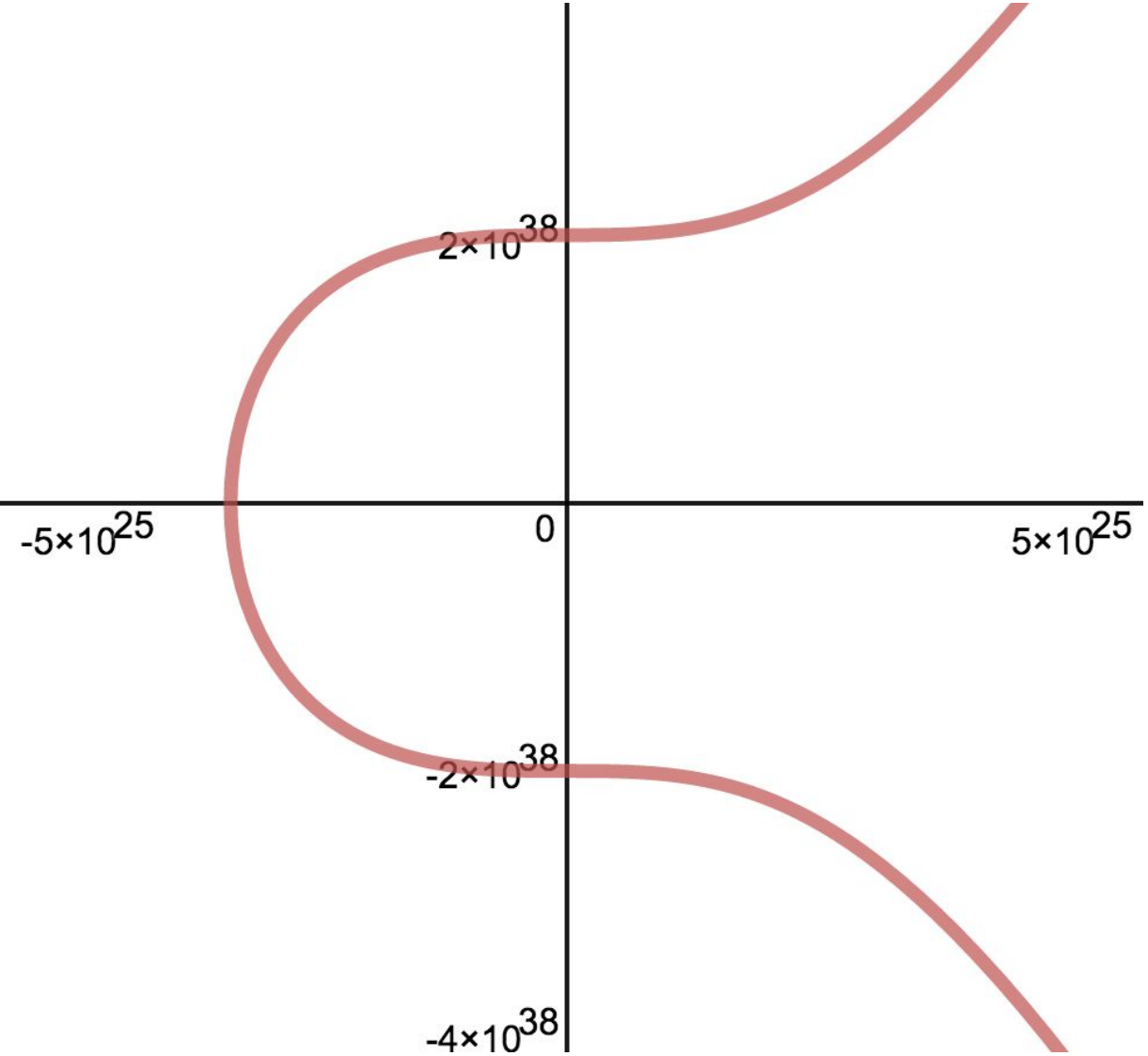
Copyright (c) 2015 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

New draft on elliptic curves sent to RFC editor.

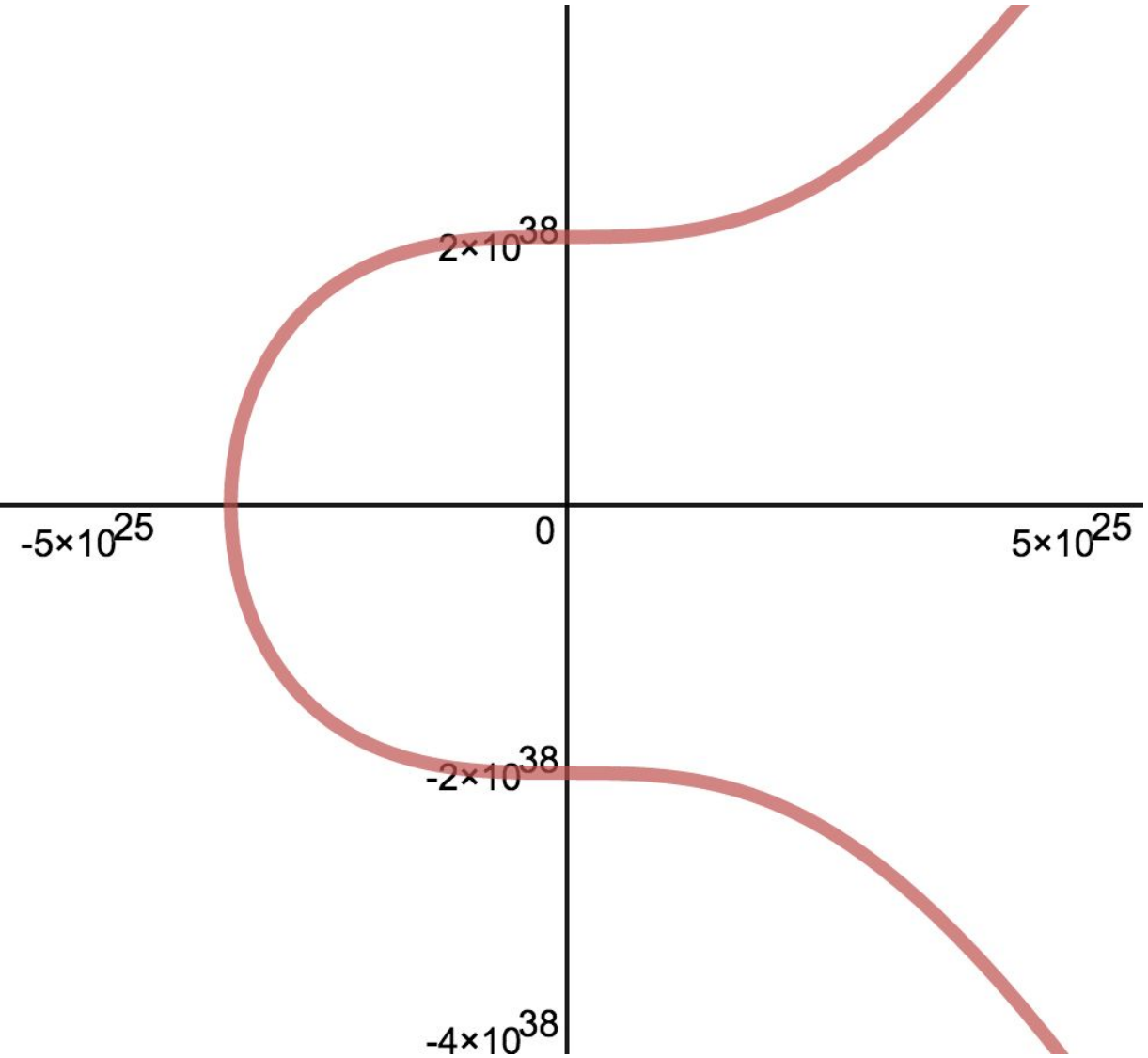
October 2015

Weierstrass curves



P-256: $y^2 = x^3 - 3x + b$

$b = 41058363725152142129326129780047268409114441015993725554835256314039467401291$



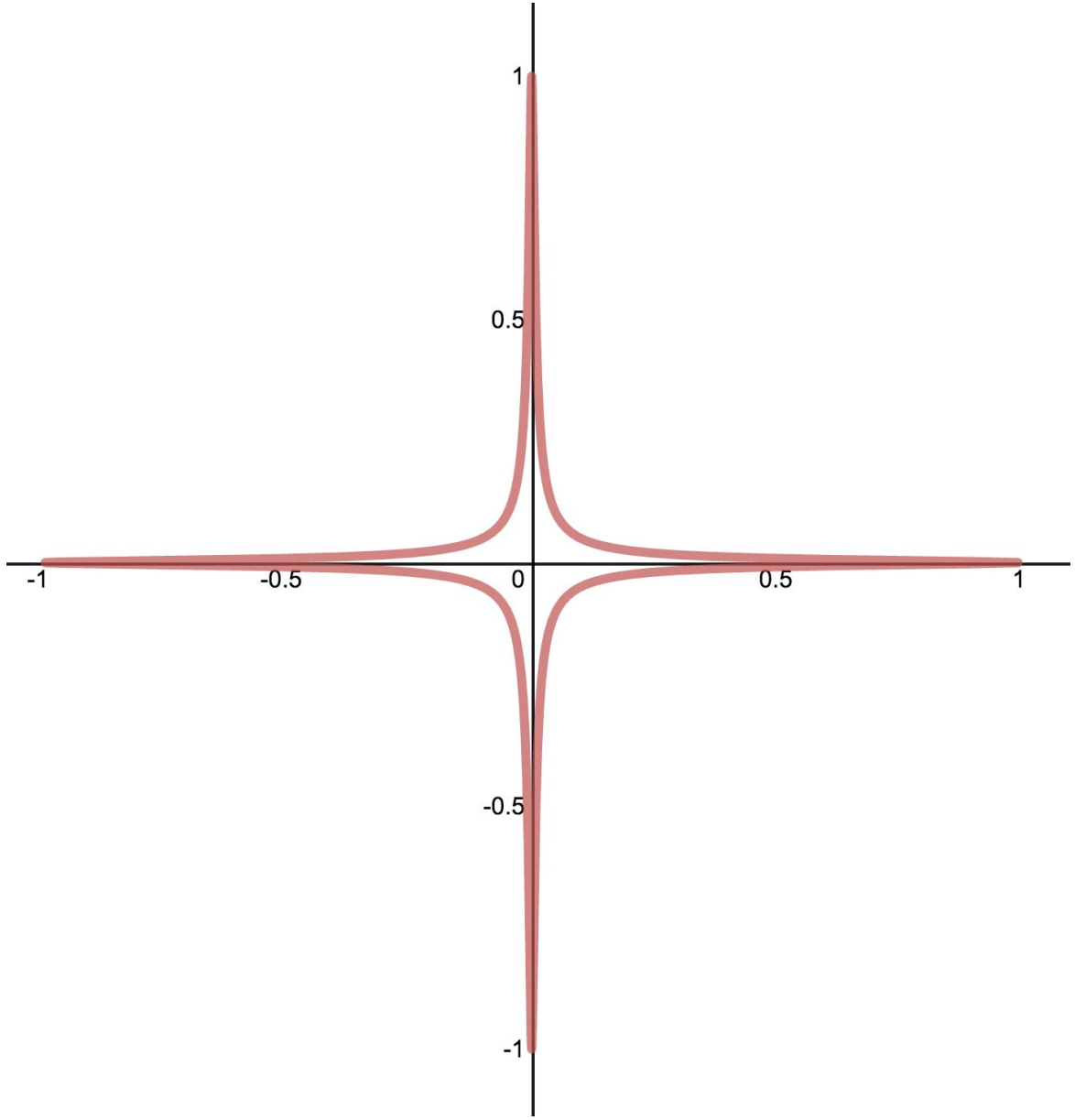
P-256: $y^2 = x^3 - 3x + b$

$b = 41058363725152142129326129780047268409114441015993725554835256314039467401291$

Weierstrass curves

- *If x_1 is not x_2: $(x_1,y_1)+(x_2,y_2) = (x_3,y_3)$ where $x_3 = \lambda^2-x_1-x_2$, $y_3 = \lambda(x_1-x_3)-y_1$, $\lambda=(y_2-y_1)/(x_2-x_1)$.*
- *If y_1 is not 0: $(x_1,y_1)+(x_1,y_1) = (x_3,y_3)$ where $x_3 = \lambda^2-x_1-x_1$, $y_3 = \lambda(x_1-x_3)-y_1$, $\lambda=(3x_1^2-3)/2y_1$.*
- *$(x_1,y_1)+(x_1,-y_1) = \infty$.*
- *$(x_1,y_1) + \infty = (x_1,y_1)$.*
- *$\infty + (x_2,y_2) = (x_2,y_2)$.*
- *$\infty + \infty = \infty$.*

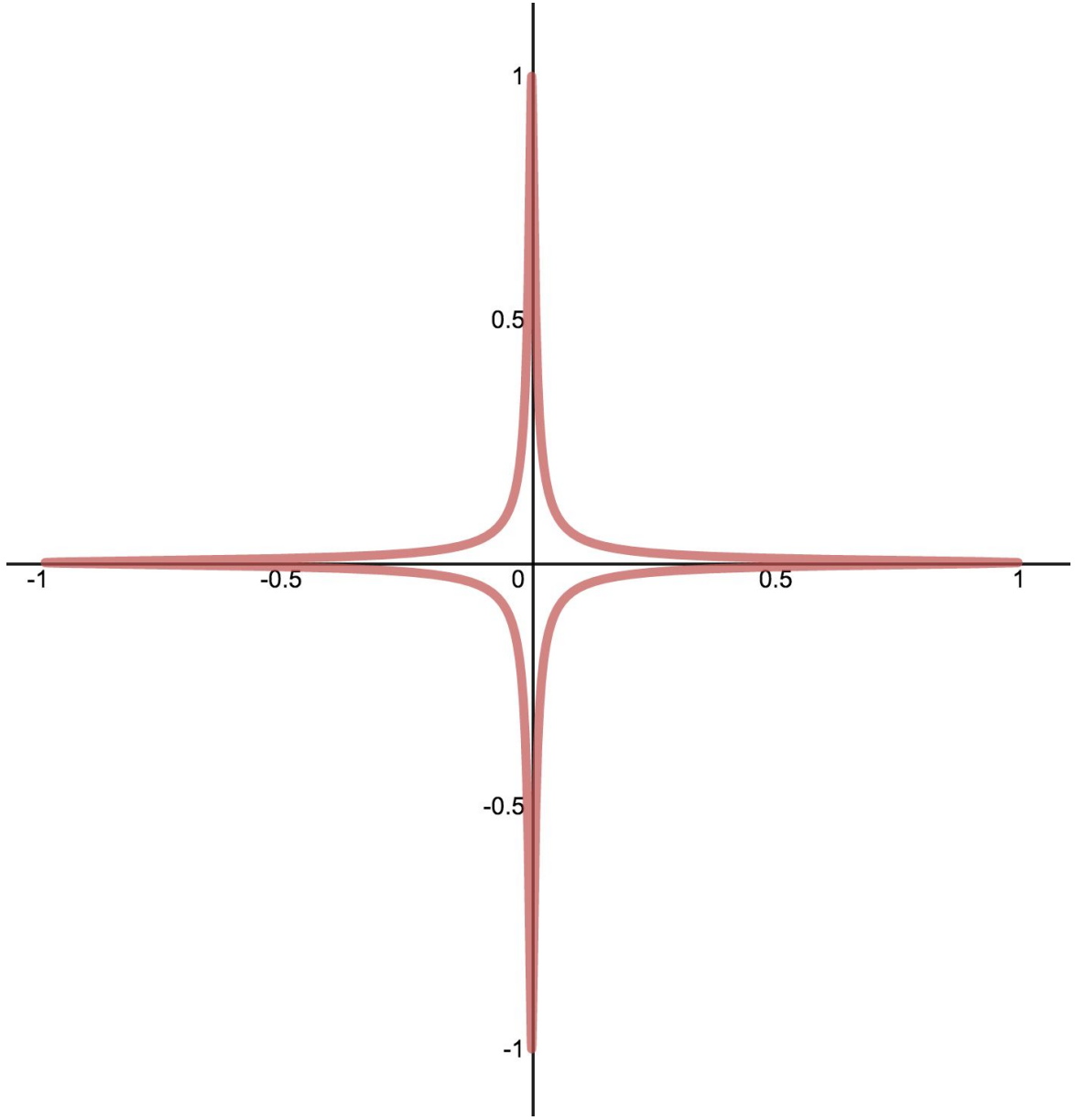
Edwards curves



Ed448: $x^2 + y^2 = x^3 + 1 - 39081x^2y^2$

Edwards curves

$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1 y_2 + y_1 x_2}{1 + d x_1 x_2 y_1 y_1}, \frac{y_1 y_2 - x_1 x_2}{1 - d x_1 x_2 y_1 y_1} \right)$$



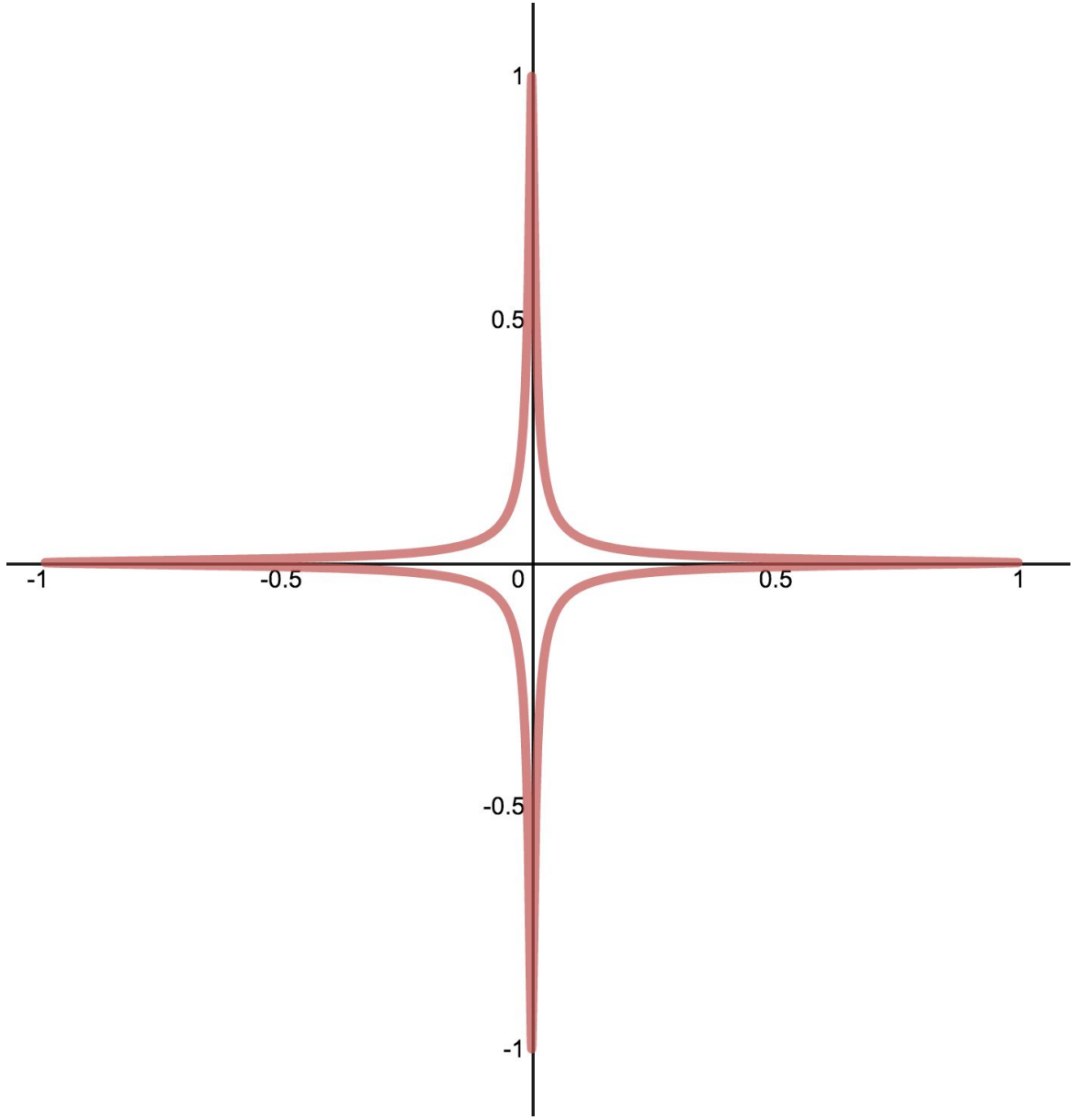
Ed448: $x^2 + y^2 = x^3 + 1 - 39081 x^2 y^2$

Edwards curves

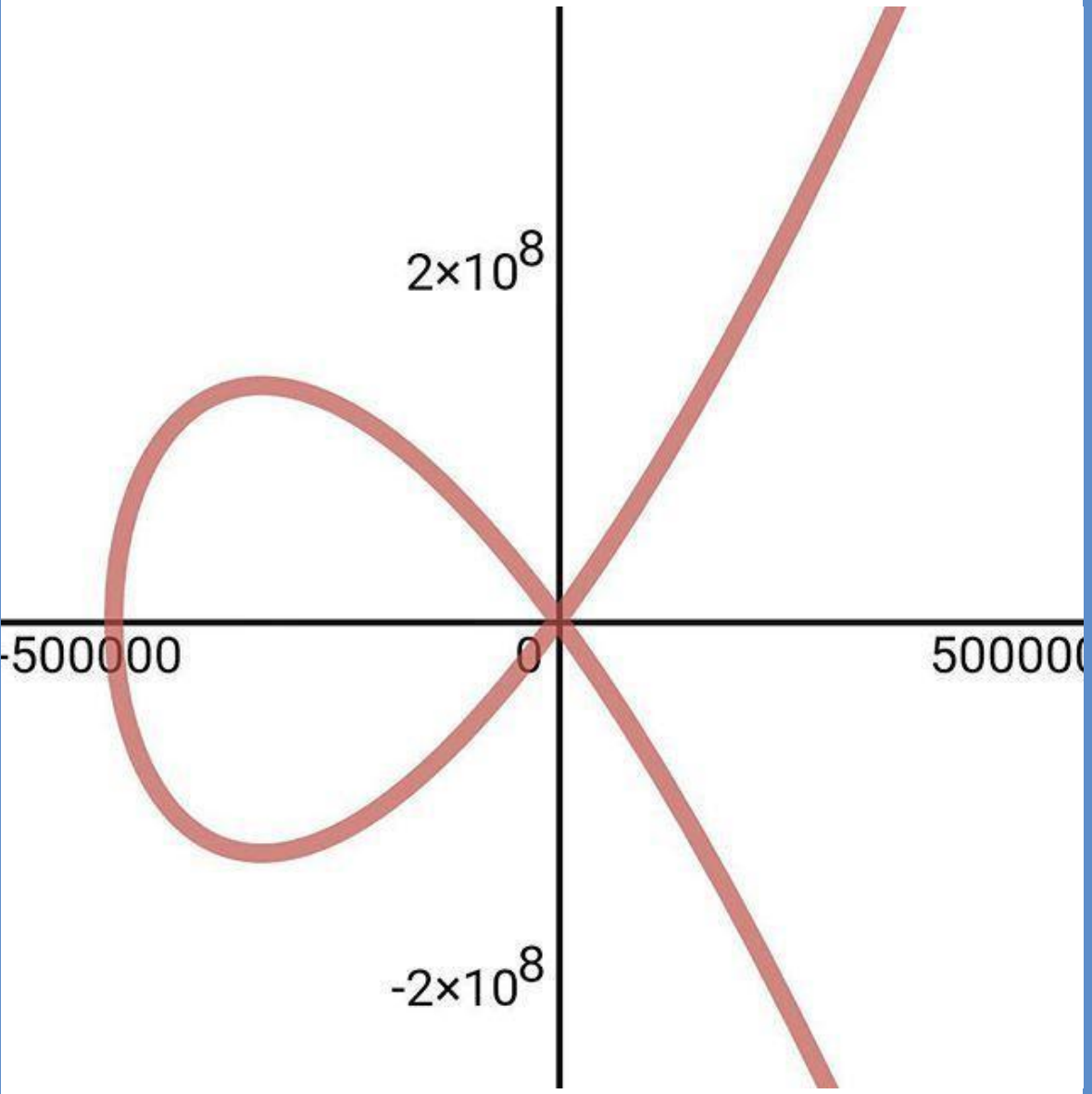
$$(x_1, y_1) + (x_2, y_2) = \left(\frac{x_1 y_2 + y_1 x_2}{1 + d x_1 x_2 y_1 y_1}, \frac{y_1 y_2 - x_1 x_2}{1 - d x_1 x_2 y_1 y_1} \right)$$

Single, complete addition law

Faster, easier to implement, few edge cases, does not compute for invalid points



Ed448: $x^2 + y^2 = x^3 + 1 - 39081 x^2 y^2$



Curve25519: $y^2 = x^3 + 486662x^2 + x$

RFC 7768

Published January 2016

Curve25519, Curve448

TLS 1.3, X.509 certificates, more

Backdoored RNG
New IETF Standards
Post-Quantum Crypto

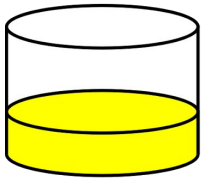
Backdoored RNG
New IETF Standards
Post-Quantum Crypto

Shor's Algorithm

Diffie-Hellman Key Exchange

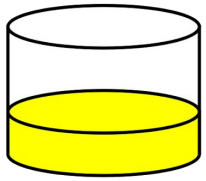
Diffie-Hellman Key Exchange

Alice



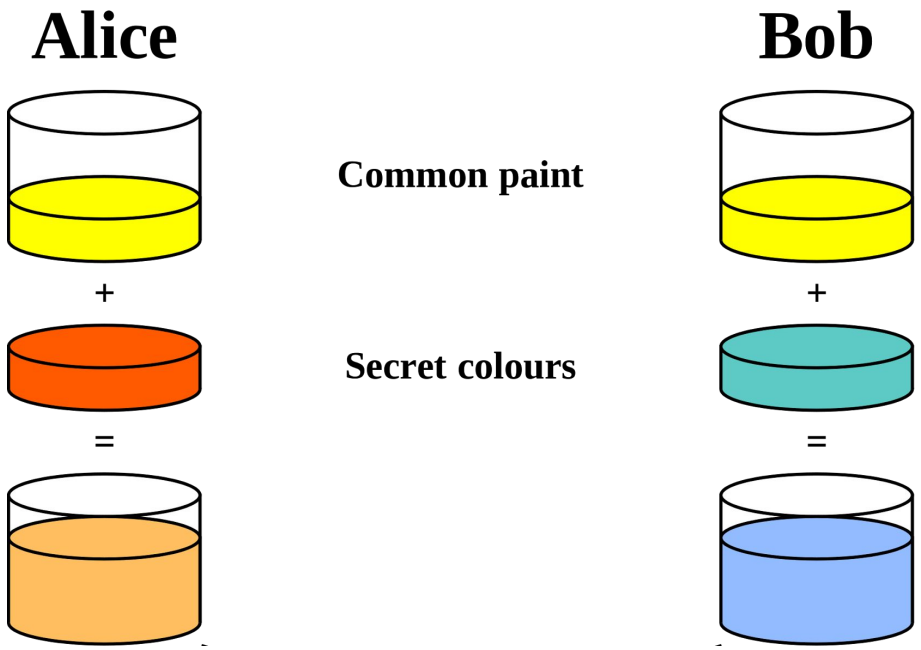
Common paint

Bob

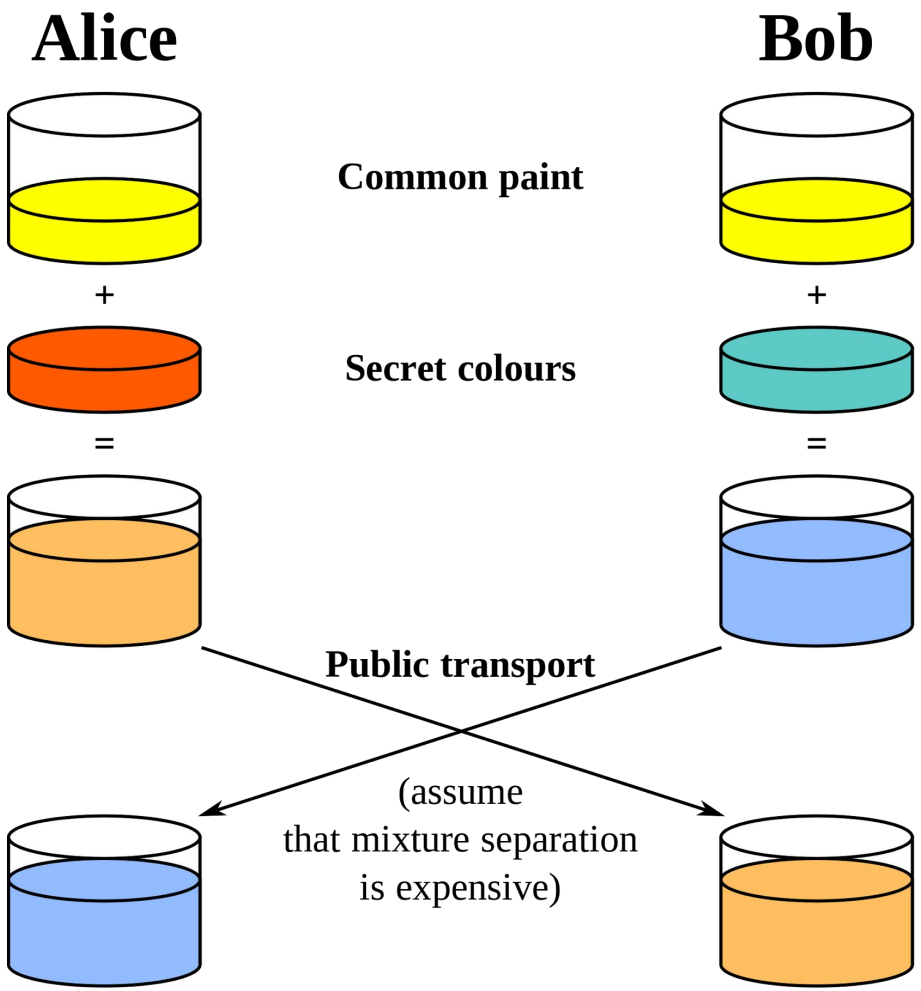


Diffie-Hellman Key Exchange

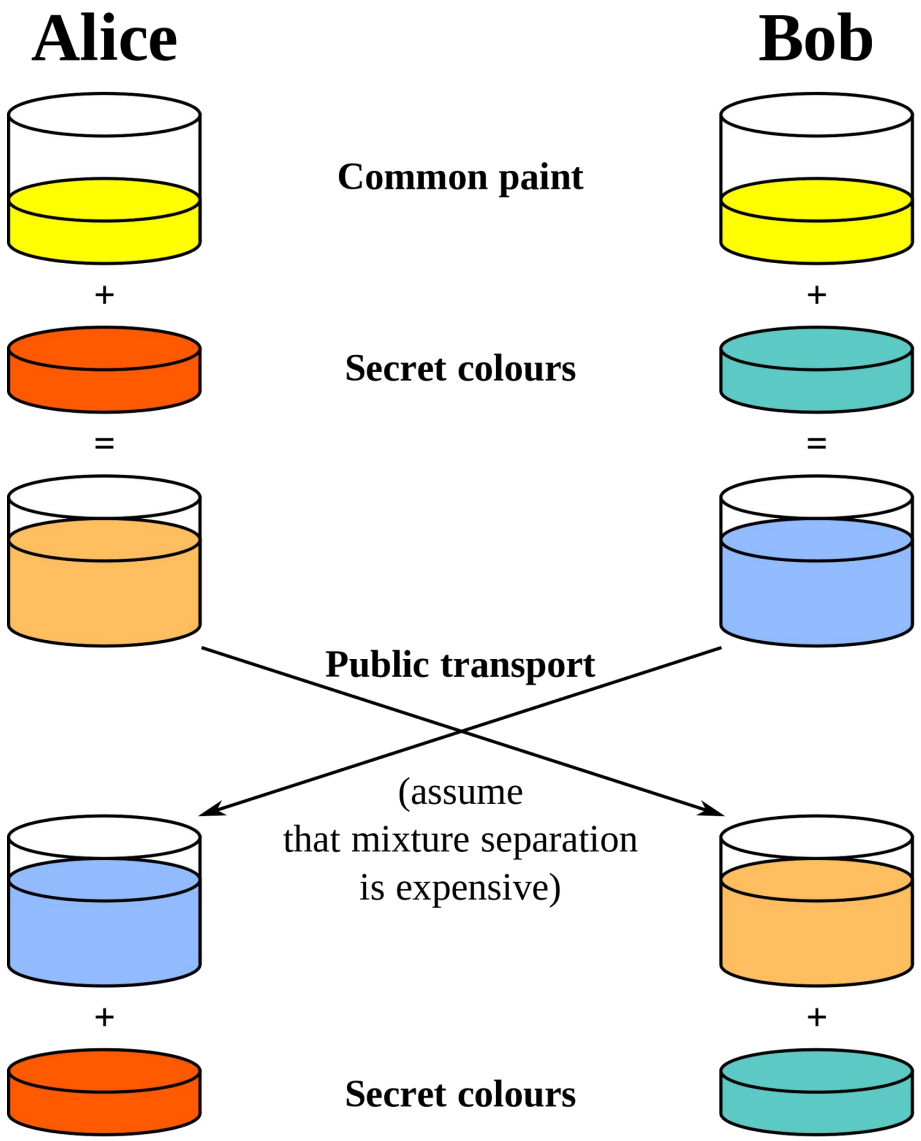
Diffie-Hellman Key Exchange



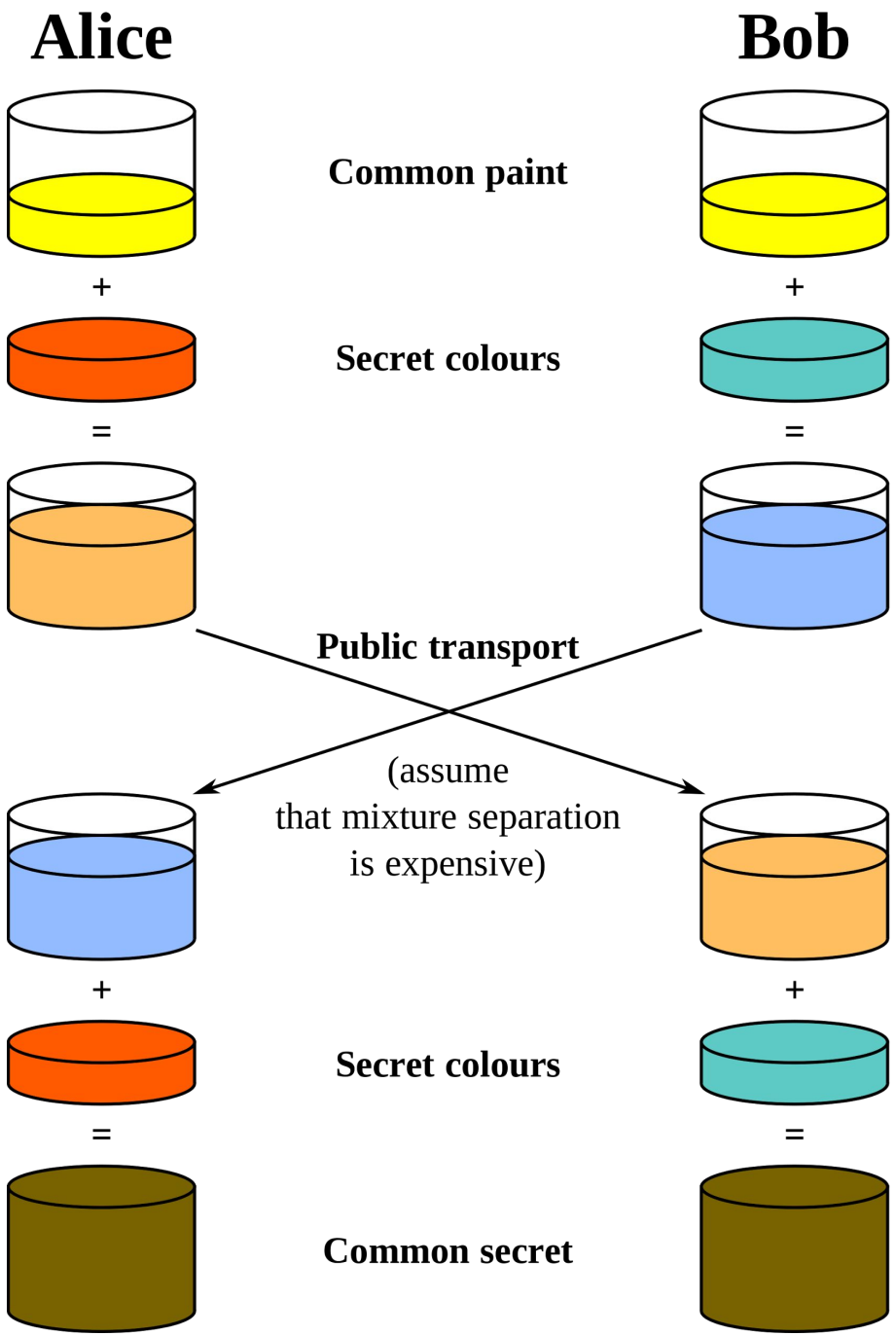
Diffie-Hellman Key Exchange

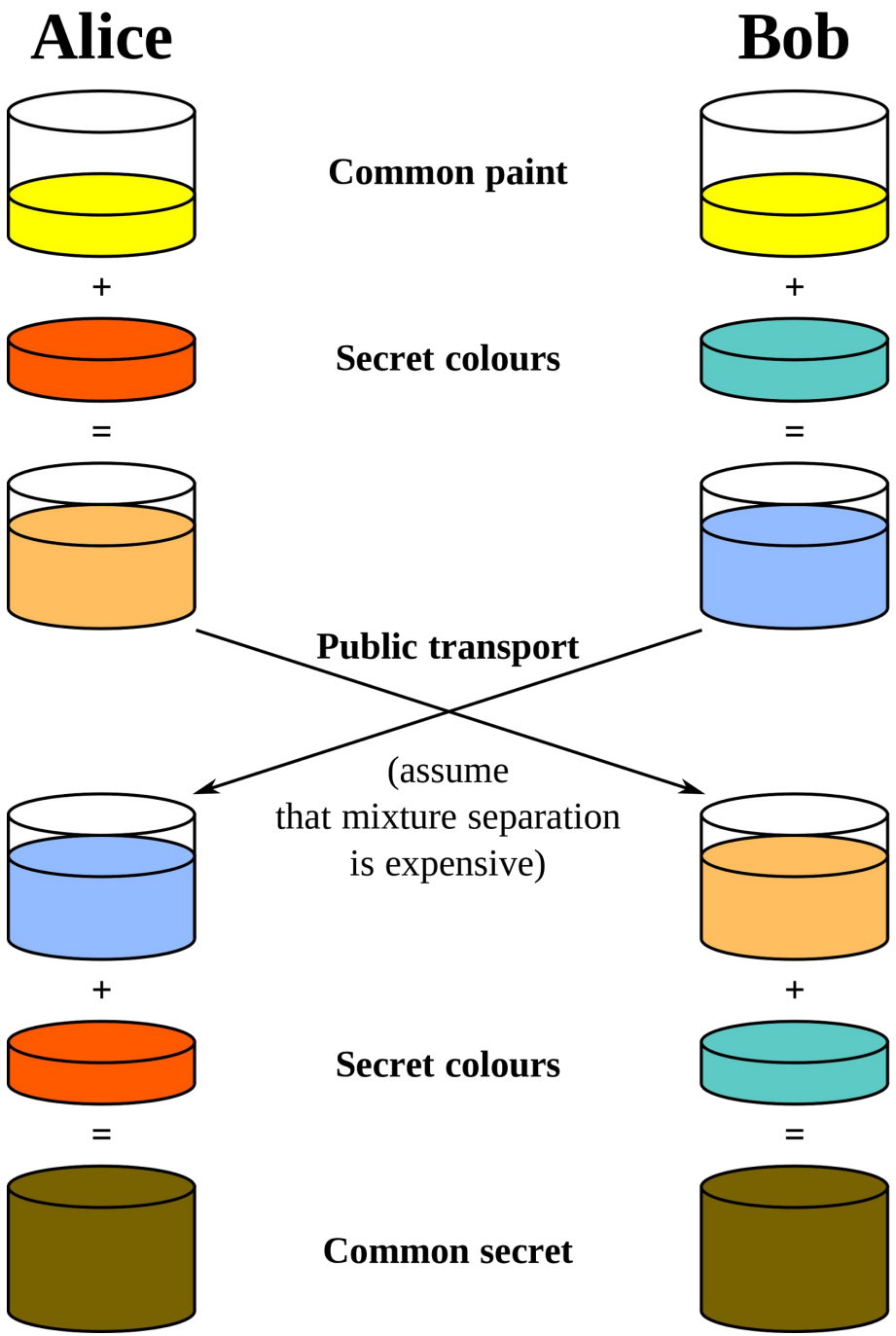


Diffie-Hellman Key Exchange



Diffie-Hellman Key Exchange

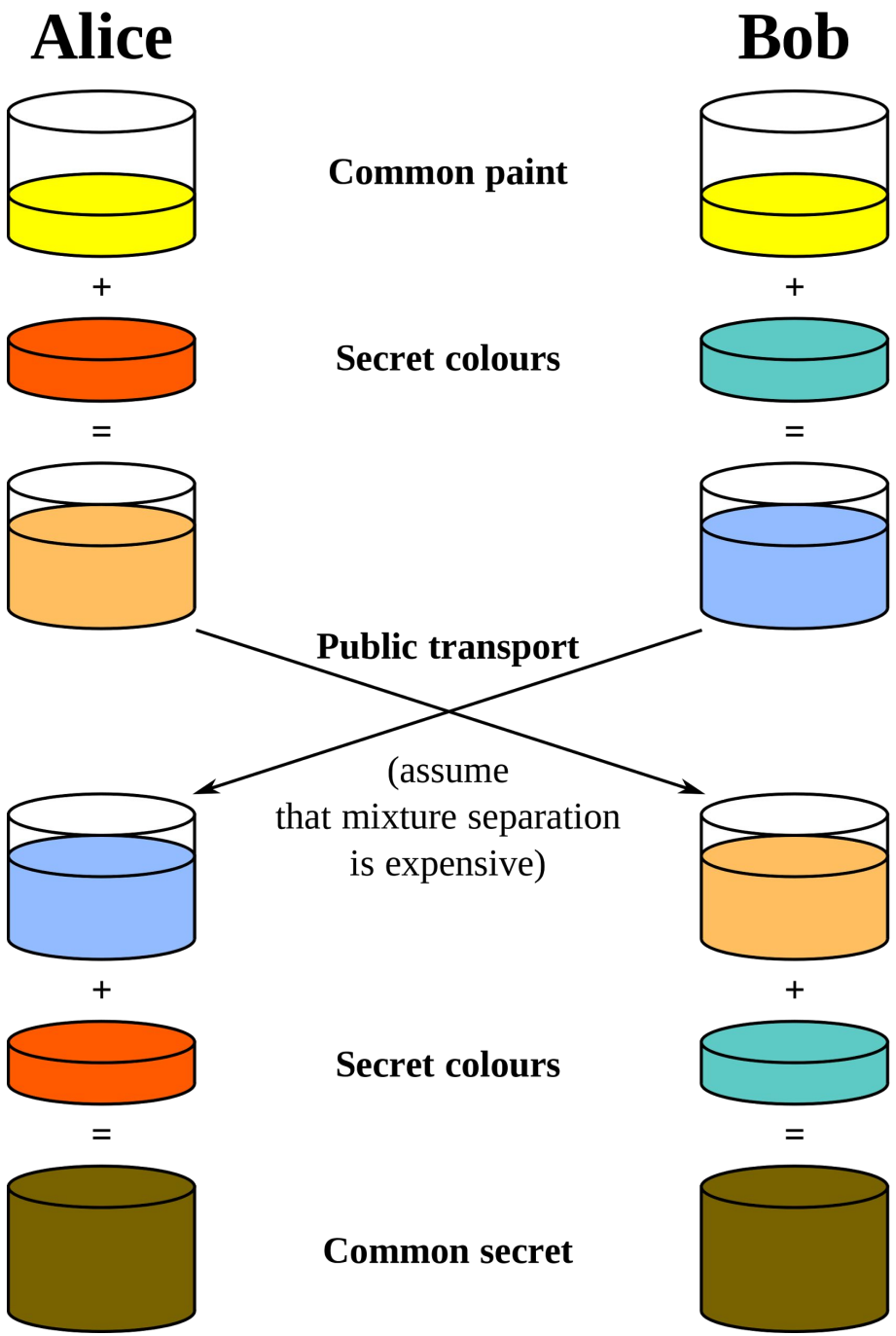




Diffie-Hellman Key Exchange

Originally **FFDH:** $key = g^{ab} \bmod p$

Exchange **integers**



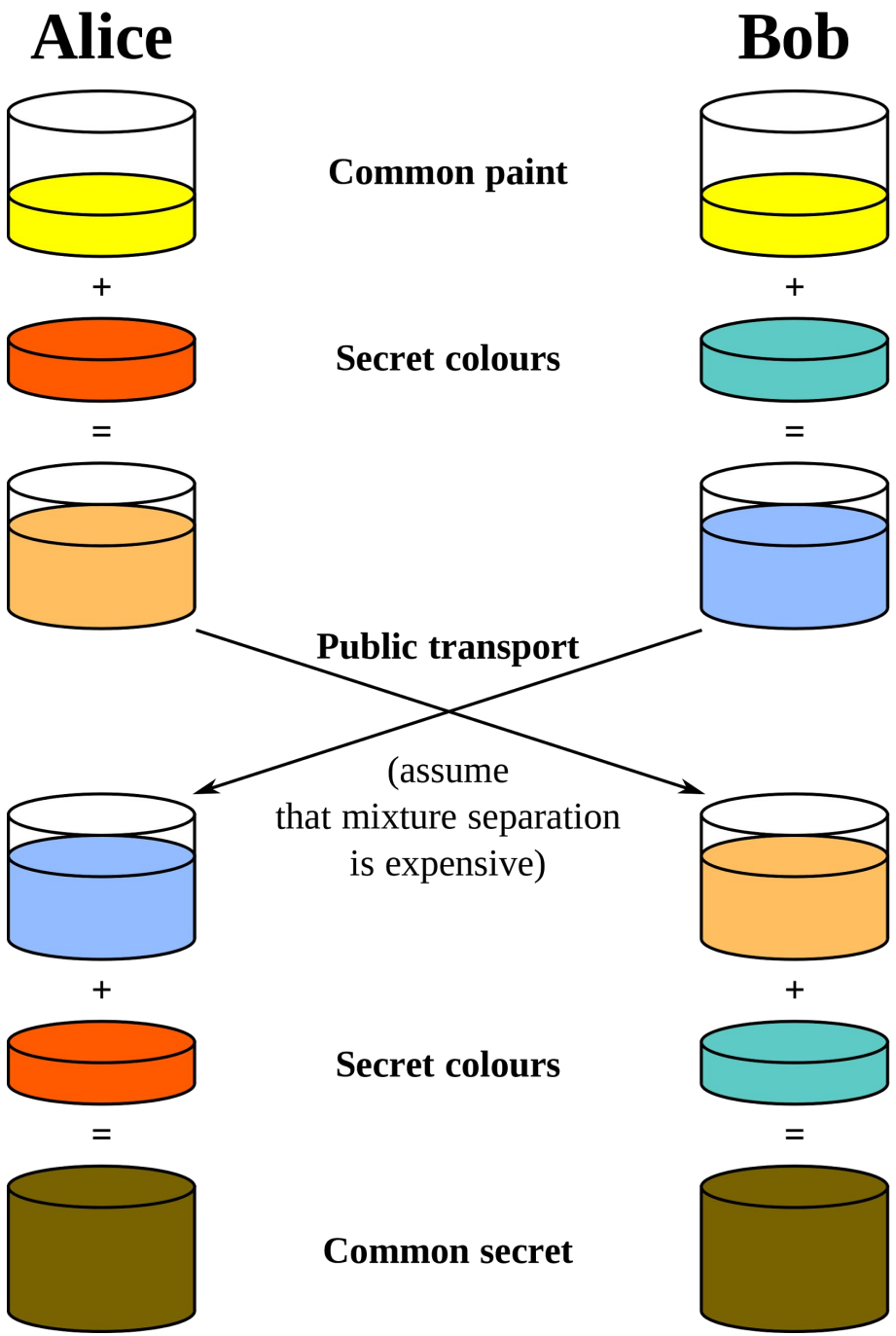
Diffie-Hellman Key Exchange

Originally **FFDH:** $key = g^{ab} \bmod p$

Exchange **integers**

Then **ECDH:** $key = abP \bmod p$

Exchange **points** on curve



Diffie-Hellman Key Exchange

Originally **FFDH:** $key = g^{ab} \bmod p$

Exchange **integers**

Then **ECDH:** $key = abP \bmod p$

Exchange **points** on curve

Now **SIDH:**

$$key = \phi'_a(\phi_b(E)) = \phi'_B(\phi_A(E))$$

Exchange whole **curves**

$$\phi : E \to E'$$

$$\phi : E \to E'$$

$$\phi(x, y) = (x', y')$$

$$\phi(x, y) = -(x, y)$$

$$\phi(x, y) = -(x, y)$$

$$\phi(\mathcal{O}) = -(\mathcal{O}) = \mathcal{O}$$

$$\phi(x, y) = -(x, y)$$

$$\phi(\mathcal{O}) = -(\mathcal{O}) = \mathcal{O}$$

$$\#E = \#E'$$

$$\phi(x,y) = \left(\frac{x^{372} + x\ldots}{x^{279} + \ldots}, y \frac{x^{279} + x\ldots}{x^{372} + x\ldots} \right)$$

$$\phi_n = \phi_{n-1} \circ \cdots \circ \phi_0$$

Alice

E

Bob

E

Alice

$$E$$

$$+$$

$$\phi_A = f(E, n_A)$$

Bob

$$E$$

$$+$$

$$\phi_B = f(E, n_B)$$

Alice

$$E$$

$$+$$

$$\phi_A = f(E, n_A)$$

$$=$$

$$E_A = \phi_A(E)$$

Bob

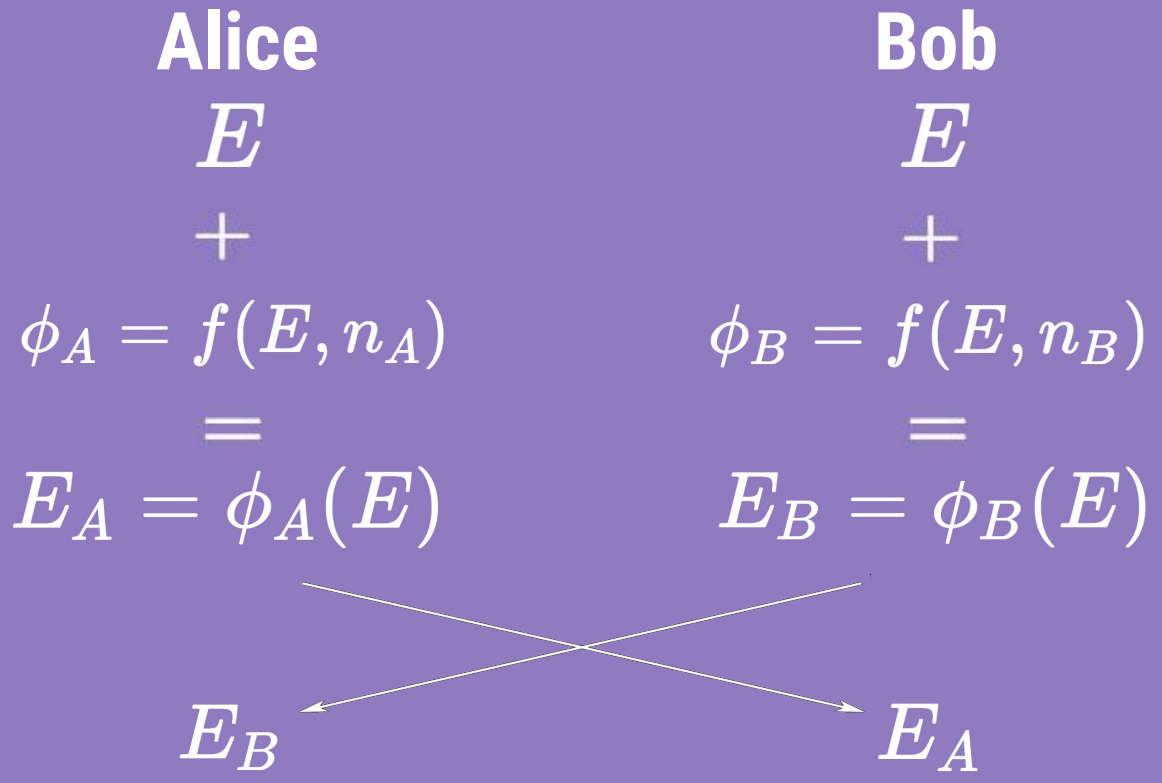
$$E$$

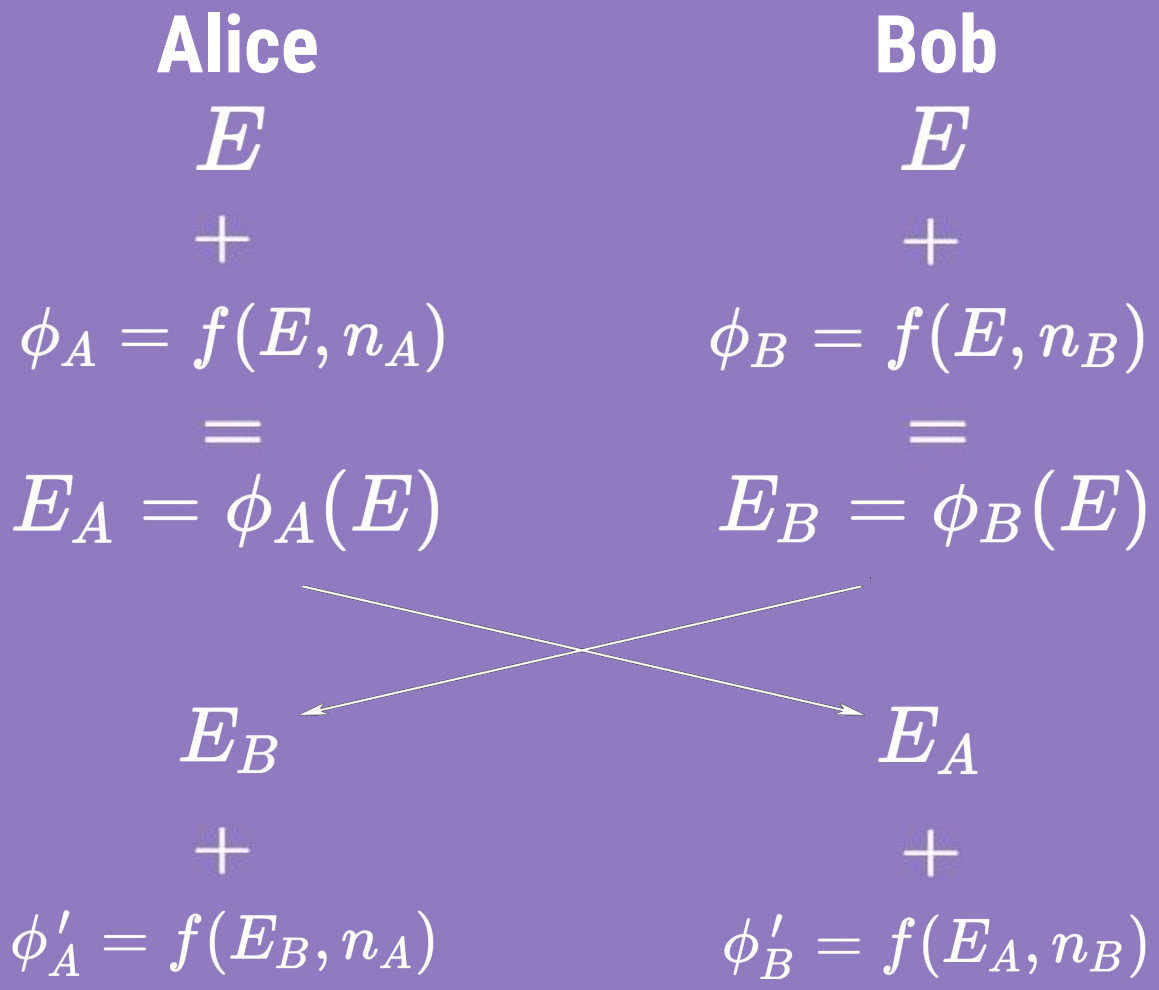
$$+$$

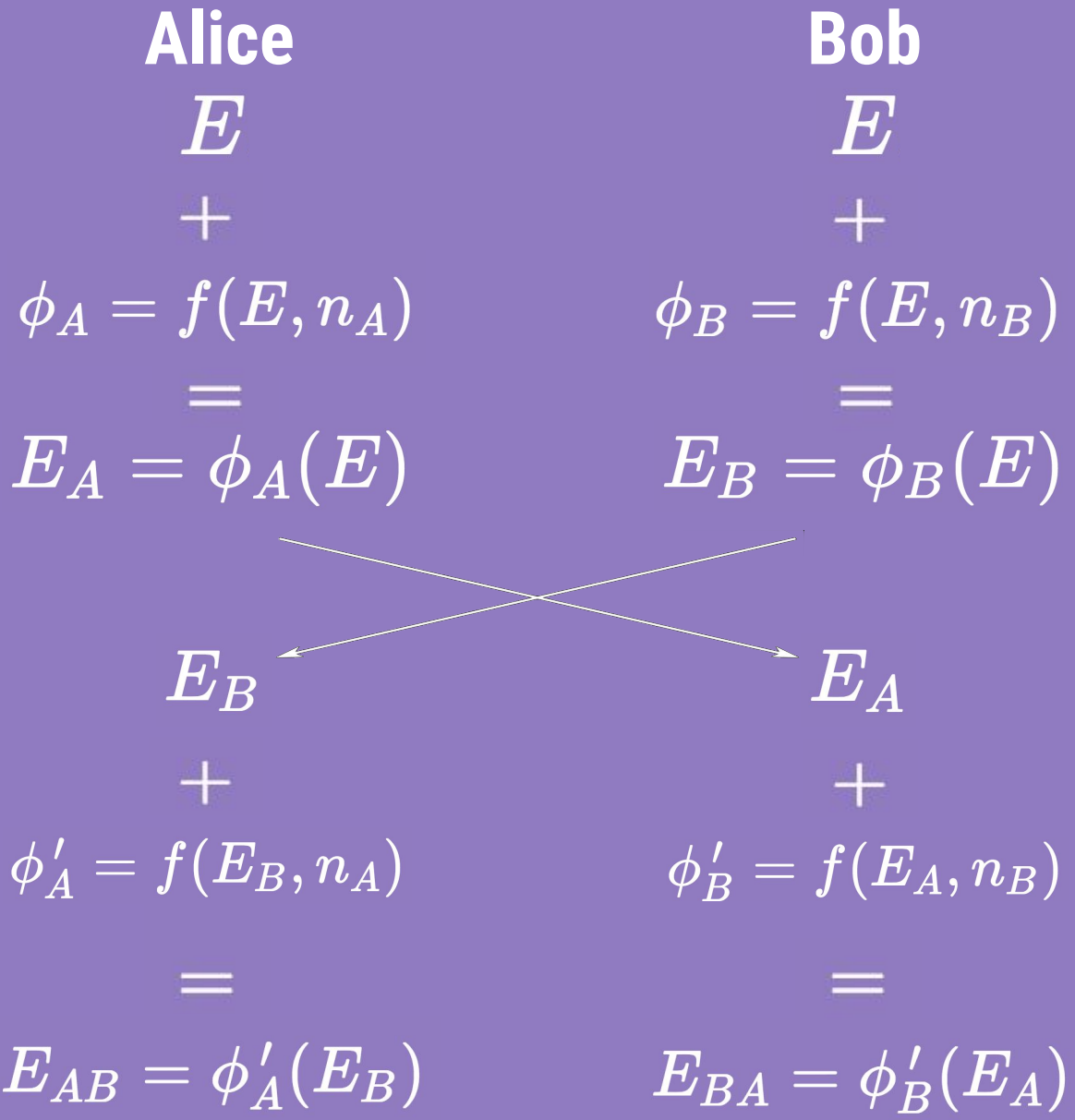
$$\phi_B = f(E, n_B)$$

$$=$$

$$E_B = \phi_B(E)$$


Alice
Bob
E
E
$+$
$+$
$\phi_A = f(E, n_A)$
$\phi_B = f(E, n_B)$
$=$
$=$
$E_A = \phi_A(E)$
$E_B = \phi_B(E)$
E_B
E_A


Alice
E
+
$\phi_A = f(E, n_A)$
=
$E_A = \phi_A(E)$
Bob
E
+
$\phi_B = f(E, n_B)$
=
$E_B = \phi_B(E)$
E_B
+
$\phi'_A = f(E_B, n_A)$
E_A
+
$\phi'_B = f(E_A, n_B)$


Alice
Bob
E
E
$+$
$+$
$\phi_A = f(E, n_A)$
$\phi_B = f(E, n_B)$
$=$
$=$
$E_A = \phi_A(E)$
$E_B = \phi_B(E)$
E_B
E_A
$+$
$+$
$\phi'_A = f(E_B, n_A)$
$\phi'_B = f(E_A, n_B)$
$=$
$=$
$E_{AB} = \phi'_A(E_B)$
$E_{BA} = \phi'_B(E_A)$

Alice	Bob
E	E
$+$	$+$
$\phi_A = f(E, n_A)$	$\phi_B = f(E, n_B)$
$=$	$=$
$E_A = \phi_A(E)$	$E_B = \phi_B(E)$
E_B	E_A
$+$	$+$
$\phi'_A = f(E_B, n_A)$	$\phi'_B = f(E_A, n_B)$
$=$	$=$
$E_{AB} = \phi'_A(E_B)$	$E_{BA} = \phi'_B(E_A)$

$$j(E_{AB}) = j(E_{BA})$$

Efficient algorithms for supersingular isogeny Diffie-Hellman

Craig Costello, Patrick Longa, and Michael Naehrig

Microsoft Research, USA

Abstract. We propose a new suite of algorithms that significantly improve the performance of supersingular isogeny Diffie-Hellman (SIDH) key exchange. Subsequently, we present a full-fledged implementation of SIDH that is geared towards the 128-bit quantum and 192-bit classical security levels. Our library is the first constant-time SIDH implementation and is up to 2.9 times faster than the previous best (non-constant-time) SIDH software. The high speeds in this paper are driven by compact, inversion-free point and isogeny arithmetic and fast SIDH-tailored field arithmetic: on an Intel Haswell processor, generating ephemeral public keys takes 46 million cycles for Alice and 52 million cycles for Bob, while computing the shared secret takes 44 million and 50 million cycles, respectively. The size of public keys is only 564 bytes, which is significantly smaller than most of the popular post-quantum key exchange alternatives. Ultimately, the size and speed of our software illustrates the strong potential of SIDH as a post-quantum key exchange candidate and we hope that these results encourage a wider cryptanalytic effort.

Keywords: Post-quantum cryptography, Diffie-Hellman key exchange, supersingular elliptic curves, isogenies, SIDH.

Table 2. Performance results (expressed in millions of clock cycles) of the proposed SIDH implementation in comparison with the implementation by Azarderakhsh et al. [2] on x64 platforms. Benchmark tests were taken with Intel's TurboBoost disabled and the results were rounded to the nearest 10^6 clock cycles. Benchmarks were done on a 3.4GHz Intel Core i7-2600 Sandy Bridge and a 3.4GHz Intel Core i7-4770 Haswell processor running Ubuntu 14.04 LTS.

Operation	This work		Prior work [2]	
	Sandy Bridge	Haswell	Sandy Bridge	Haswell
Alice's keygen	50	46	165	149
Bob's keygen	57	52	172	152
Alice's shared key	47	44	133	118
Bob's shared key	55	50	137	122
Total	207	192	608	540

Remark 7. In Section 4 we discussed several specialized choices that were made for reasons unrelated to performance, e.g., in the name of simplicity and/or compactness. We stress that, should future cryptanalysis reveal that these choices introduce a security vulnerability, the performance of SIDH and the performance improvements in Sections 3 and 5 are unlikely to be affected (in any meaningful way) by reverting back to the more general case(s). In particular, if it turns out that sampling from a fraction of the possible 2- and 3-torsion subgroups gives an attacker some appreciable advantage, then modifying the code to sample from the full set of torsion subgroups is merely an exercise, and the subsequent performance difference would be unnoticeable. Similarly, if any of (i) starting on a subfield curve (see Remark 2), (ii) using of the base-field and trace-zero subgroups, or (iii) using the distortion map, turns out to degrade SIDH security, then the main upshot of reverting to randomized public generators or starting on a curve minimally defined over $\mathbb{F}_{p^2}$ would be the inflated public parameters (see Section 6); the slowdown during key generation would be minor and the shared secret computations would be unchanged.

8 BigMont: a strong ECDH+SIDH hybrid

Table 2. Performance results (expressed in millions of clock cycles) of the proposed SIDH implementation in comparison with the implementation by Azarderakhsh et al. [2] on x64 platforms. Benchmark tests were taken with Intel's TurboBoost disabled and the results were rounded to the nearest 10^6 clock cycles. Benchmarks were done on a 3.4GHz Intel Core i7-2600 Sandy Bridge and a 3.4GHz Intel Core i7-4770 Haswell processor running Ubuntu 14.04 LTS.

Operation	This work		Prior work [2]	
	Sandy Bridge	Haswell	Sandy Bridge	Haswell
Alice's keygen	50	46	165	149
Bob's keygen	57	52	172	152
Alice's shared key	47	44	133	118
Bob's shared key	55	50	137	122
Total	207	192	608	540

2.9X!

Remark 7. In Section 4 we discussed several specialized choices that were made for reasons unrelated to performance, e.g., in the name of simplicity and/or compactness. We stress that, should future cryptanalysis reveal that these choices introduce a security vulnerability, the performance of SIDH and the performance improvements in Sections 3 and 5 are unlikely to be affected (in any meaningful way) by reverting back to the more general case(s). In particular, if it turns out that sampling from a fraction of the possible 2- and 3-torsion subgroups gives an attacker some appreciable advantage, then modifying the code to sample from the full set of torsion subgroups is merely an exercise, and the subsequent performance difference would be unnoticeable. Similarly, if any of (i) starting on a subfield curve (see Remark 2), (ii) using of the base-field and trace-zero subgroups, or (iii) using the distortion map, turns out to degrade SIDH security, then the main upshot of reverting to randomized public generators or starting on a curve minimally defined over $\mathbb{F}_{p^2}$ would be the inflated public parameters (see Section 6); the slowdown during key generation would be minor and the shared secret computations would be unchanged.

8 BigMont: a strong ECDH+SIDH hybrid

Remark 7. In Section 4 we discussed several specialized choices that were made for reasons unrelated to performance, e.g., in the name of simplicity and/or compactness. We stress that, should future cryptanalysis reveal that these choices introduce a security vulnerability, the performance of SIDH and the performance improvements in Sections 3 and 5 are unlikely to be affected (in any meaningful way) by reverting back to the more general case(s). In particular, if it turns out that sampling from a fraction of the possible 2- and 3-torsion subgroups gives an attacker some appreciable advantage, then modifying the code to sample from the full set of torsion subgroups is merely an exercise, and the subsequent performance difference would be unnoticeable. Similarly, if any of (i) starting on a subfield curve (see Remark 2), (ii) using of the base-field and trace-zero subgroups, or (iii) using the distortion map, turns out to degrade SIDH security, then the main upshot of reverting to randomized public generators or starting on a curve minimally defined over $\mathbb{F}_{p^2}$ would be the inflated public parameters (see Section 6); the slowdown during key generation would be minor and the shared secret computations would be unchanged.

8 BigMont: a strong ECDH+SIDH hybrid

We now return to the discussion (from Section 1) of a hybrid scheme. Put simply, and in regards to both security and suitability, at present there is not enough confidence and consensus within the PQC community to warrant the standalone deployment of one particular post-quantum key exchange primitive. Subsequently, there is interest (cf. [6]) in deploying classical primitives alongside post-quantum primitives in order to hedge one's bets until a confidence-inspiring PQC key exchange standard arrives. This is particularly interesting in the case of SIDH, whose security has (because of its relatively short lifespan) received less cryptanalytic scrutiny than its post-quantum counterparts.

In this section we discuss how traditional ECDH key exchange can be included alongside SIDH key exchange at the price of a very small overhead. The main benefit of our approach is its simplicity; while SIDH could be partnered with ECDH on any of the standardized elliptic curves, this would mean that a lot more code needs to be written and/or maintained. In particular, it is often the case that the bulk of the code in high-speed ECC implementations relates to the underlying field arithmetic. Given that none of the fields underlying the standardized curves are SIDH-friendly[13], such a partnership would require either a generic implementation that would be much less efficient, or two unrelated implementations of field arithmetic. Our proposal avoids this additional complexity by performing ECDH on an elliptic curve defined over the same ground field as the one used for SIDH.

For $p = 2^{372}3^{239} - 1$, recall that our SIDH software works with isogenous curves $E_a/\mathbb{F}_{p^2} : y^2 = x^3 + ax^2 + x$ whose group orders are of the form $\#E_a = 2^i \cdot 3^j$, meaning that elliptic curve discrete

Remark 7. In Section 4 we discussed several specialized choices that were made for reasons unrelated to performance, e.g., in the name of simplicity and/or compactness. We stress that, should future cryptanalysis reveal that these choices introduce a security vulnerability, the performance of SIDH and the performance improvements in Sections 3 and 5 are unlikely to be affected (in any meaningful way) by reverting back to the more general case(s). In particular, if it turns out that sampling from a fraction of the possible 2- and 3-torsion subgroups gives an attacker some appreciable advantage, then modifying the code to sample from the full set of torsion subgroups is merely an exercise, and the subsequent performance difference would be unnoticeable. Similarly, if any of (i) starting on a subfield curve (see Remark 2), (ii) using of the base-field and trace-zero subgroups, or (iii) using the distortion map, turns out to degrade SIDH security, then the main upshot of reverting to randomized public generators or starting on a curve minimally defined over $\mathbb{F}_{p^2}$ would be the inflated public parameters (see Section 6); the slowdown during key generation would be minor and the shared secret computations would be unchanged.

8 BigMont: a strong ECDH+SIDH hybrid

We now return to the discussion (from Section 1) of a hybrid scheme. Put simply, and in regards to both security and suitability, at present there is not enough confidence and consensus within the PQC community to warrant the standalone deployment of one particular post-quantum key exchange primitive. Subsequently, there is interest (cf. [6]) in deploying classical primitives alongside post-quantum primitives in order to hedge one's bets until a confidence-inspiring PQC key exchange standard arrives. This is particularly interesting in the case of SIDH, whose security has (because of its relatively short lifespan) received less cryptanalytic scrutiny than its post-quantum counterparts.

In this section we discuss how traditional ECDH key exchange can be included alongside SIDH key exchange at the price of a very small overhead. The main benefit of our approach is its simplicity; while SIDH could be partnered with ECDH on any of the standardized elliptic curves, this would mean that a lot more code needs to be written and/or maintained. In particular, it is often the case that the bulk of the code in high-speed ECC implementations relates to the underlying field arithmetic. Given that none of the fields underlying the standardized curves are SIDH-friendly[13], such a partnership would require either a generic implementation that would be much less efficient, or two unrelated implementations of field arithmetic. Our proposal avoids this additional complexity by performing ECDH on an elliptic curve defined over the same ground field as the one used for SIDH.

For $p = 2^{372}3^{239} - 1$, recall that our SIDH software works with isogenous curves $E_a/\mathbb{F}_{p^2} : y^2 = x^3 + ax^2 + x$ whose group orders are of the form $\#E_a = 2^i \cdot 3^j$, meaning that elliptic curve discrete

		SIDH	SIDH+ECDH
≈ bit-security (hard problem)	classical	192 (SSDDH)	384 (ECDHP)
	PQ	128 (SSDDH)	128 (SSDDH)
public key size		564	658
speed (cc $\times 10^6$)	Alice's keygen	46	52
	Bob's keygen	52	58
	Alice's shared key	44	50
	Bob's shared key	50	57

In Table 3 we compare hybrid SIDH+ECDH versus standalone SIDH. The take-away message is that for a less than 1.17x increase in public key sizes and less than 1.13x increase in the overall computing cost, we can increase the classical security of the key exchange from 192 bits (based on the relatively new SSDDH problem) to 384 bits (based on the long-standing ECDLP).

9 Validating public keys

Recall from Section 2 that De Feo, Jao and Plût [18] prove that SIDH is *session-key secure* (under SSDDH) in the authenticated-links adversarial model [12]. This model assumes perfectly authenticated links which effectively forces adversaries to be passive eavesdroppers; in particular, it assumes that public keys are correctly generated by honest users. While this model can be suitable for key exchange protocols that are instantiated in a truly ephemeral way, in real-world scenarios it is often the case that (static) private keys are reused. This can incentivize malicious users to create faulty public keys that allow them to learn information about the other user's static private key, and in such scenarios validating public keys becomes a mandatory practical requirement.

In traditional elliptic curve Diffie-Hellman (ECDH), validating public keys essentially amounts to checking that points are on the correct and cryptographically secure curve [8]. Such *point validation* is considered trivial in ECDH, since checking that a point satisfies a curve equation requires only a handful of field multiplications and additions, and this is negligible compared to the overall cost (e.g., of a scalar multiplication).

In contexts where SIDH private keys are reused, public key validation is equally as important but is no longer as trivial. In April 2015, a group from the NSA [28] pointed out that "direct public key validation is not always possible for [...] isogeny based schemes" before describing more complicated options that validate public keys *indirectly*. In this section we describe ways to directly validate various properties of our public keys that, in particular, work entirely in our compact framework, i.e., without the need of y-coordinates or of the Montgomery b coefficient that fixes the quadratic twist.

Recall from Section 6 that an honest user generates public keys of the form

		SIDH	SIDH+ECDH
≈ bit-security (hard problem)	classical	192 (SSDDH)	384 (ECDHP)
	PQ	128 (SSDDH)	128 (SSDDH)
public key size		564	658
speed (cc $\times 10^6$)	Alice's keygen	46	52
	Bob's keygen	52	58
	Alice's shared key	44	50
	Bob's shared key	50	57

In Table 3 we compare hybrid SIDH+ECDH versus standalone SIDH. The take-away message is that for a less than 1.17x increase in public key sizes and less than 1.13x increase in the overall computing cost, we can increase the classical security of the key exchange from 192 bits (based on the relatively new SSDDH problem) to 384 bits (based on the long-standing ECDLP).

9 Validating public keys

Recall from Section 2 that De Feo, Jao and Plût [18] prove that SIDH is *session-key secure* (under SSDDH) in the authenticated-links adversarial model [12]. This model assumes perfectly authenticated links which effectively forces adversaries to be passive eavesdroppers; in particular, it assumes that public keys are correctly generated by honest users. While this model can be suitable for key exchange protocols that are instantiated in a truly ephemeral way, in real-world scenarios it is often the case that (static) private keys are reused. This can incentivize malicious users to create faulty public keys that allow them to learn information about the other user's static private key, and in such scenarios validating public keys becomes a mandatory practical requirement.

In traditional elliptic curve Diffie-Hellman (ECDH), validating public keys essentially amounts to checking that points are on the correct and cryptographically secure curve [8]. Such *point validation* is considered trivial in ECDH, since checking that a point satisfies a curve equation requires only a handful of field multiplications and additions, and this is negligible compared to the overall cost (e.g., of a scalar multiplication).

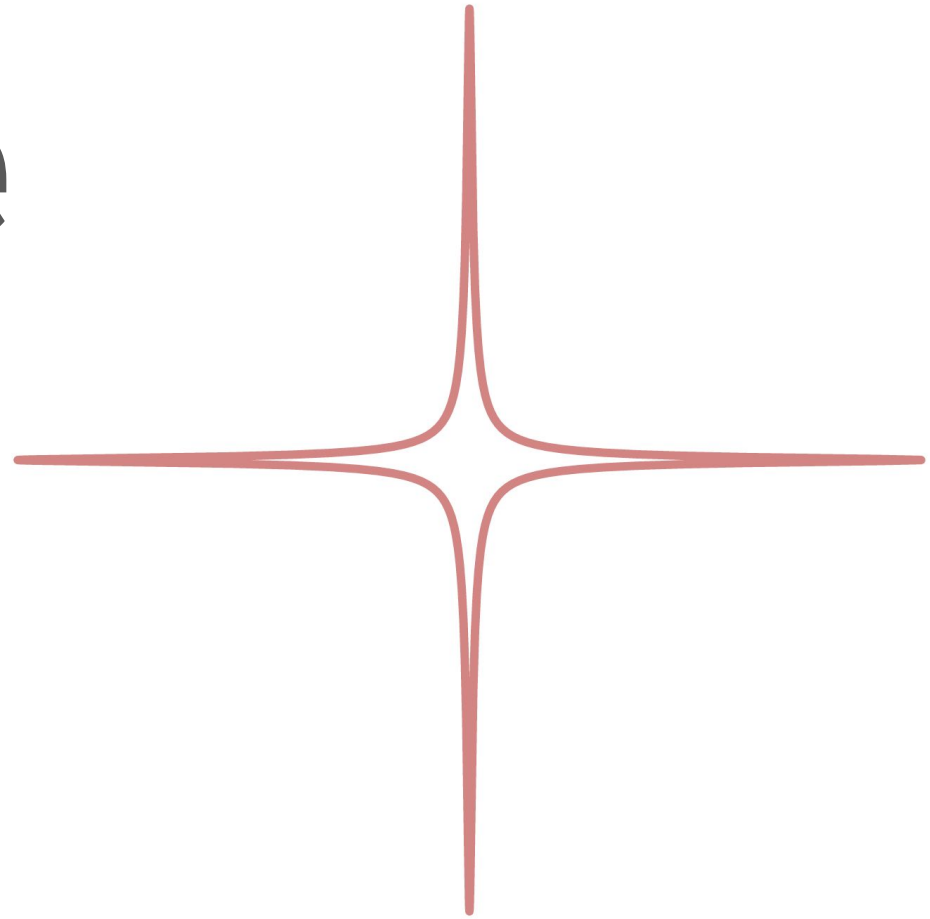
In contexts where SIDH private keys are reused, public key validation is equally as important but is no longer as trivial. In April 2015, a group from the NSA [28] pointed out that "direct public key validation is not always possible for [...] isogeny based schemes" before describing more complicated options that validate public keys *indirectly*. In this section we describe ways to directly validate various properties of our public keys that, in particular, work entirely in our compact framework, i.e., without the need of y-coordinates or of the Montgomery b coefficient that fixes the quadratic twist.

Recall from Section 6 that an honest user generates public keys of the form

Backdoored RNG
New IETF Standards
Post-Quantum Crypto

State of the Curve

DEFCON 2016



Deirdre Connolly
@durumcrustulum