Lecture 10 – Diffie-Hellman III, PKE, KEM, ElGamal, Noise

TEK4500

21.10.2026

Håkon Jacobsen

hakon.jacobsen@its.uio.no

Basic goals of cryptography

	Message privacy	**Message integrity / authentication**	
Symmetric keys	Symmetric encryption	Message authentication codes (MAC)	
Asymmetric keys	Asymmetric encryption (a.k.a. public-key encryption)	Digital signatures	Key exchange

Public-key encryption



Public-key encryption (PKE) – syntax

A **PKE** is a tuple $\mathrm{PKE} = (\mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec})$ of algorithms

$\mathrm{KeyGen} : \mathcal{SK} \times \mathcal{PK}$

$(sk, pk) \twoheadleftarrow \mathrm{KeyGen}$

$\mathrm{Enc} : \mathcal{PK} \times \mathcal{M} \to \mathcal{C}$

$\mathrm{Enc}(pk, M) = C$

$\mathrm{Dec} : \mathcal{SK} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$

$\mathrm{Dec}(sk, C) = M/\perp$

Correctness: $\forall (sk, pk) \twoheadleftarrow \mathrm{KeyGen}$ and $M \in \mathcal{M}$:

$$\mathrm{Dec}\big(sk, \mathrm{Enc}(pk, M)\big) = M$$

PKE-security: IND-CPA

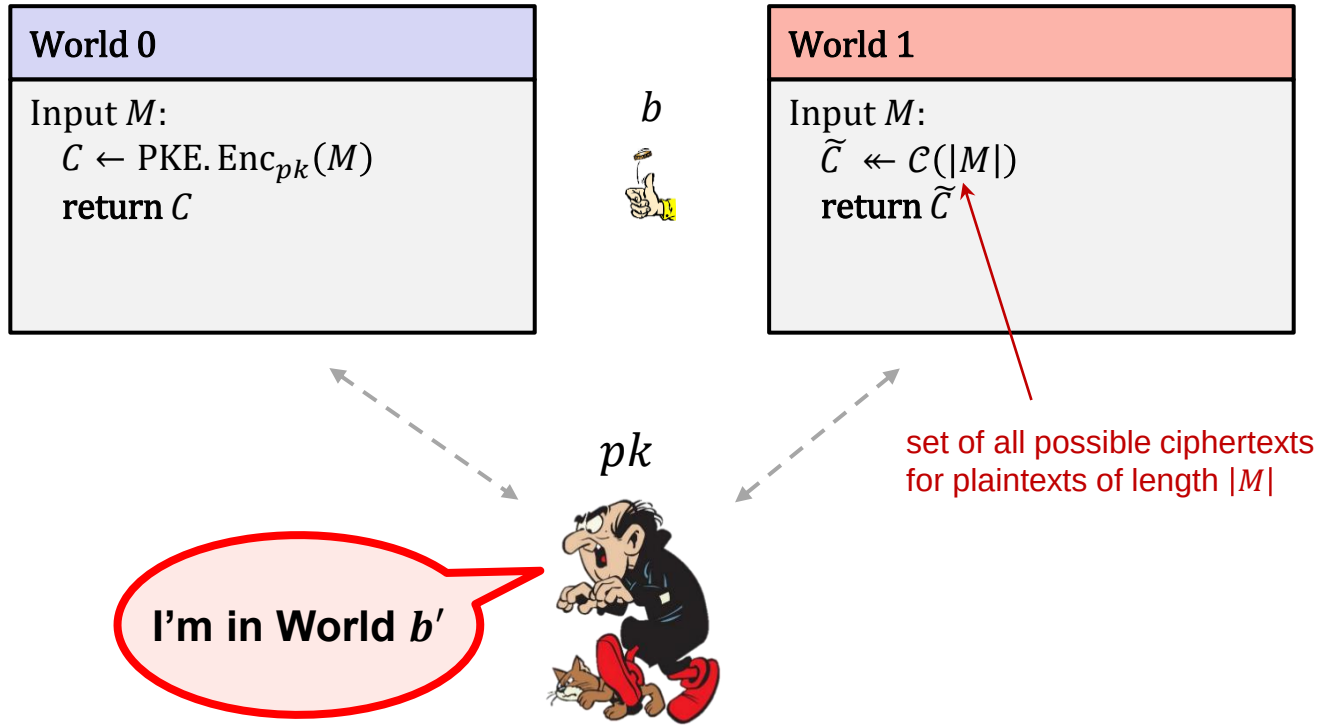
$\mathbf{Exp}^{\text{ind-cpa}}_{\text{PKE}}(A)$

1. $b \twoheadleftarrow \{0,1\}$
2. $(sk, pk) \twoheadleftarrow \text{PKE.KeyGen}$
3. $b' \leftarrow A^{\mathcal{E}}(pk)$
4. **return** $b' \stackrel{?}{=} b$

$\mathcal{E}(M)$

---

1. $C_0 \leftarrow \text{PKE.Enc}_{pk}(M)$
2. $C_1 \twoheadleftarrow \mathcal{C}(|M|)$
3. **return** C_b



Definition: The **IND-CPA advantage** of A is

$$\mathbf{Adv}^{\text{ind-cpa}}_{\text{PKE}}(A) \stackrel{\text{def}}{=} 2 \cdot |\Pr[b' = b] - 1/2|$$

Public-key encryption: ElGamal

$G = \langle g \rangle$



M



A

B

$a \twoheadleftarrow \{1, \ldots, |G|\}$

$A \leftarrow g^a$

$Z \leftarrow B^a$

$C \leftarrow \Sigma.\mathrm{Enc}(Z, M)$

C

$b \twoheadleftarrow \{1, \ldots, |G|\}$

$B \leftarrow g^b$

$Z \leftarrow A^b$

$M \leftarrow \Sigma.\mathrm{Dec}(Z, C)$

Symmetric encryption scheme

Public-key encryption: ElGamal

$H : \boldsymbol{G} \to \{0,1\}^k$

$G = \langle g \rangle , B$

KeyGen

1. $sk = b \twoheadleftarrow \{1, \dots, |\boldsymbol{G}|\}$
2. $pk = B \leftarrow g^b$
3. return (sk, pk)

M



B

A , C



Enc(pk, M)

1. $a \twoheadleftarrow \{1, \dots, |\boldsymbol{G}|\}$
2. $A \leftarrow g^a$
3. $Z \leftarrow B^a$
4. $C \leftarrow \Sigma.\mathrm{Enc}(Z, M)$
5. return (A, C)

ElGamal IND-CPA security:

C

- DLOG + DH must be hard
- Σ must be IND-CPA secure
- Is this enough?

Dec$(sk, (A, C))$

1. $Z \leftarrow A^b$
2. $M \leftarrow \Sigma.\mathrm{Dec}(Z, C)$
3. return M

$\Sigma.\mathrm{Enc} : G \times \mathcal{M} \to \mathcal{C}$

actually want $\Sigma.\mathrm{Enc} : \{0,1\}^k \times \mathcal{M} \to \mathcal{C}$

- **No:** Σ only guarantees security if the key Z is *independent* and *uniformly* distributed in the group $\boldsymbol{G}$

  …but $Z = g^{ab}$ *isn't* uniformly distributed in $\boldsymbol{G}$*!*

Public-key encryption: Hashed-ElGamal

$H : \boldsymbol{G} \to \{0,1\}^k$

$G = \langle g \rangle$

M



B

A, C



KeyGen

1. $sk = b \twoheadleftarrow \{1, \dots, |\boldsymbol{G}|\}$
2. $pk = B \leftarrow g^b$
3. return (sk, pk)

Enc(pk, M)

1. $a \twoheadleftarrow \{1, \dots,$
2. $A \leftarrow g^a$
3. $Z \leftarrow H(B^a)$
4. $C \leftarrow \Sigma.\mathrm{Enc}(Z, M)$
5. return (A, C)

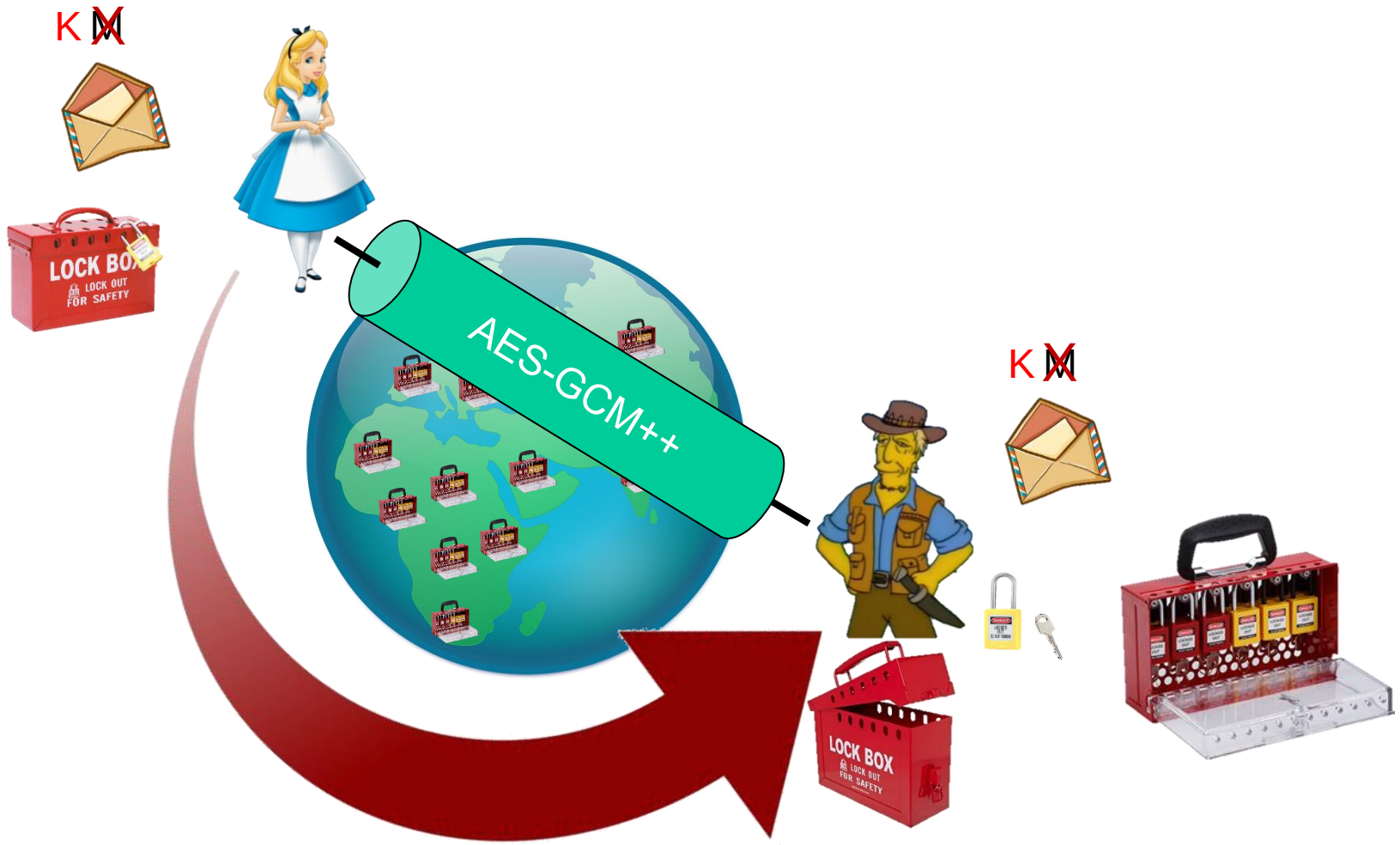
Theorem: Hashed-ElGamal is IND-CPA secure if:

1. DH-problem is hard in $\boldsymbol{G}$
2. Σ is (one-time) IND-CPA secure
3. H is "perfect" (i.e., a **random oracle**)

$C))$

1. $Z \leftarrow H(A^b)$
2. $M \leftarrow \Sigma.\mathrm{Dec}(Z, C)$
3. return M

Public-key encryption



Basic goals of cryptography

	Message privacy	**Message integrity / authentication**	
Symmetric keys	Symmetric encryption	Message authentication codes (MAC)	
Asymmetric keys	~~Asymmetric encryption (a.k.a. public-key encryption)~~	Digital signatures	Key exchange

Key Encapsulation Mechanisms (KEM)

Key Encapsulation Mechanisms (KEM) – syntax

A **KEM** is a tuple $\mathrm{KEM} = (\mathrm{KeyGen}, \mathrm{Encaps}, \mathrm{Decaps})$ of algorithms

$\mathrm{KeyGen} : \mathcal{DK} \times \mathcal{EK}$

$(dk, ek) \twoheadleftarrow \mathrm{KeyGen}$

$\mathrm{Encaps}: \mathcal{EK} \to \mathcal{K} \times \mathcal{C}$

$\mathrm{Encaps}(ek) = (K, C)$

$\mathrm{Decaps} : \mathcal{DK} \times \mathcal{C} \to \mathcal{K} \cup \{\perp\}$

$\mathrm{Decaps}(dk, C) = K/\perp$

- $\mathcal{DK}$ – private key space
- $\mathcal{EK}$ – public key space
- $\mathcal{K}$ – symmetric key space
- $\mathcal{C}$ – ciphertext space

Correctness: $\forall (dk, ek) \twoheadleftarrow \mathrm{KeyGen}$ and $(K, C) \leftarrow \mathrm{Encaps}(ek)$

$$\mathrm{Decaps}(dk, C) = K$$

KEM from PKE

KEM. KeyGen

1. $(sk, pk) \leftarrow$ PKE. KeyGen
2. **return** $(dk = sk, ek = pk)$



$pk\ (= ek)$



KEM. Encaps(pk)

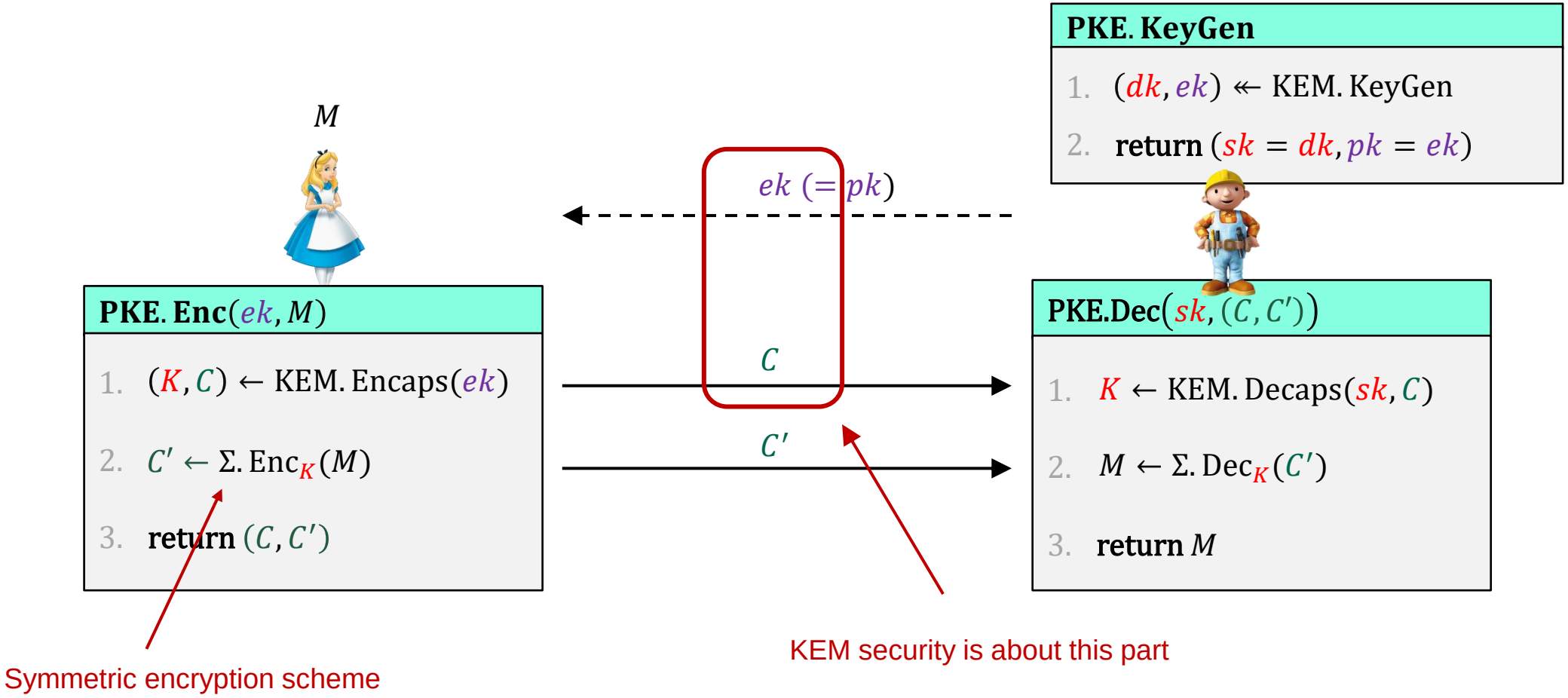
1. $K \leftarrow \mathcal{M}$
2. $C \leftarrow$ PKE. Enc(ek, K)
3. **return** (K, C)

C

KEM.Dec(dk, C)

1. $K \leftarrow$ PKE. Dec(dk, C)
2. **return** K

PKE from KEM


PKE. KeyGen
1. $(dk, ek) \leftarrow$ KEM. KeyGen
2. return $(sk = dk, pk = ek)$
M
$ek\ (= pk)$
C
C'
PKE. Enc(ek, M)
1. $(K, C) \leftarrow$ KEM. Encaps(ek)
2. $C' \leftarrow \Sigma.\mathrm{Enc}_K(M)$
3. return (C, C')
PKE.Dec$(sk, (C, C'))$
1. $K \leftarrow$ KEM. Decaps(sk, C)
2. $M \leftarrow \Sigma.\mathrm{Dec}_K(C')$
3. return M
Symmetric encryption scheme
KEM security is about this part

KEM-security: IND-CPA

$\mathbf{Exp}^{\text{ind-cpa}}_{\text{KEM}}(A)$

1. $b \twoheadleftarrow \{0,1\}$
2. $(dk, ek) \twoheadleftarrow \text{KEM.KeyGen}$
3. $b' \leftarrow A^{\mathcal{E}} \quad (ek)$
4. **return** $b' \stackrel{?}{=} b$

$\mathcal{E}()$

---

1. $(K_0, C_0) \leftarrow \text{KEM.Encaps}(ek)$
2. $K_1 \twoheadleftarrow \mathcal{K}$
3. $C_1 \twoheadleftarrow \mathcal{C}(|K|)$
4. **return** (K_b, C)

World 0

Input:
$(K, C) \leftarrow \text{KEM.Encaps}(ek)$
return (K, C)

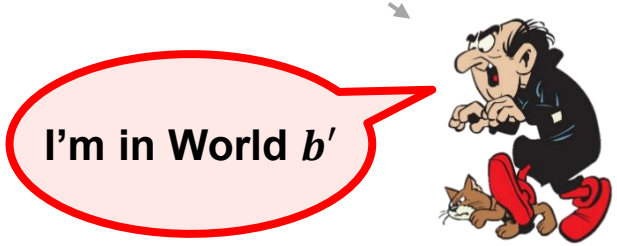
b



World 1

Input:
$\widetilde{C} \twoheadleftarrow \mathcal{C}(|K|)$
return $(\$, \widetilde{C})$

ek



Definition: The **IND-CPA advantage** of A is

$$\mathbf{Adv}^{\text{ind-cpa}}_{\text{KEM}}(A) \stackrel{\text{def}}{=} 2 \cdot |\Pr[b' = b] - 1/2|$$

KEM-security: IND-CCA

$\mathbf{Exp}^{\text{ind-cca}}_{\text{KEM}}(A)$

1. $b \twoheadleftarrow \{0,1\}$
2. Ciphertexts $\leftarrow [\,]$
3. $(dk, ek) \twoheadleftarrow \text{KEM.KeyGen}$
4. $b' \leftarrow A^{\mathcal{E}\ ,\mathcal{D}(C)}(ek)$
5. **return** $b' \stackrel{?}{=} b$

$\mathcal{E}()$

1. $(K_0, C_0) \leftarrow \text{KEM.Encaps}(ek)$
2. $K_1 \twoheadleftarrow \mathcal{K}$
3. $C_1 \twoheadleftarrow \mathcal{C}(|M|)$
4. Ciphertexts.Add(C_b)
5. **return** (K_b, C_b)

$\mathcal{D}(C)$

1. **if** $C \in$ Ciphertexts**:**
2. **return** $\perp$
3. **return** KEM.Decaps(dk, C)

World 0

Input:
$(K, C) \leftarrow \text{KEM.Encaps}(ek)$
return (K, C)

Input C:
return KEM.Decaps(dk, C)

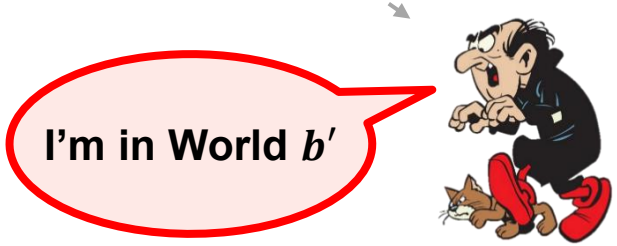
b

World 1

Input:
$\widetilde{C} \twoheadleftarrow \mathcal{C}(|K|)$
return $(\$, \widetilde{C})$

Input C:
return KEM.Decaps(dk, C)

ek



Restriction: not allowed to forward output from Encaps query to Decaps query *unmodified*

Definition: The **IND-CCA advantage** of A is

$$\mathbf{Adv}^{\text{ind-cca}}_{\text{KEM}}(A) \stackrel{\text{def}}{=} 2 \cdot |\Pr[b' = b] - 1/2|$$

Lavabit

Lavabit email service shuts down abruptly, blaming US government



Graham Cluley • @gcluley
9:44 pm, August 8, 2013



Get 12 weeks for ~~$29.99~~ $6

THE NEW YORKER

SCIENCE AND TECH Can Computers Learn Common Sense? The Ocean's Most Delicate Species

ANNALS OF TECHNOLOGY

HOW LAVABIT MELTED DOWN

By Michael Phillips and Matt Buchanan
October 7, 2013

ne Guardian
e, funded by readers
ubscribe →

Search jobs | Sign in | Search | International e

The Guardian
News website of the year

Opinion | Sport | Culture | Lifestyle | More

s Climate crisis Environment Science Global development Football **Tech** Business Obituaries

This article is more than **9 years old**

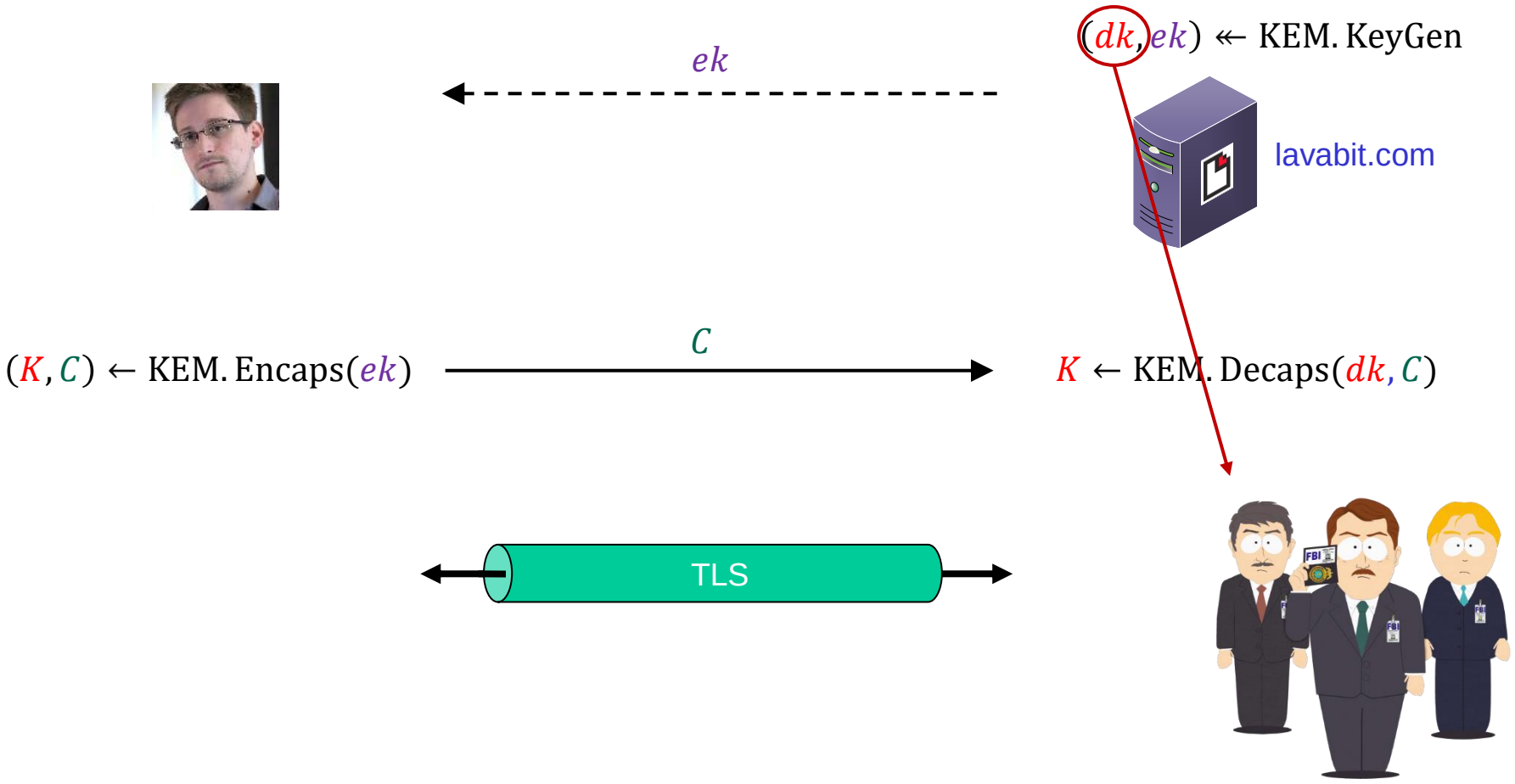
Lavabit email service abruptly shut down citing government interference

Most viewed

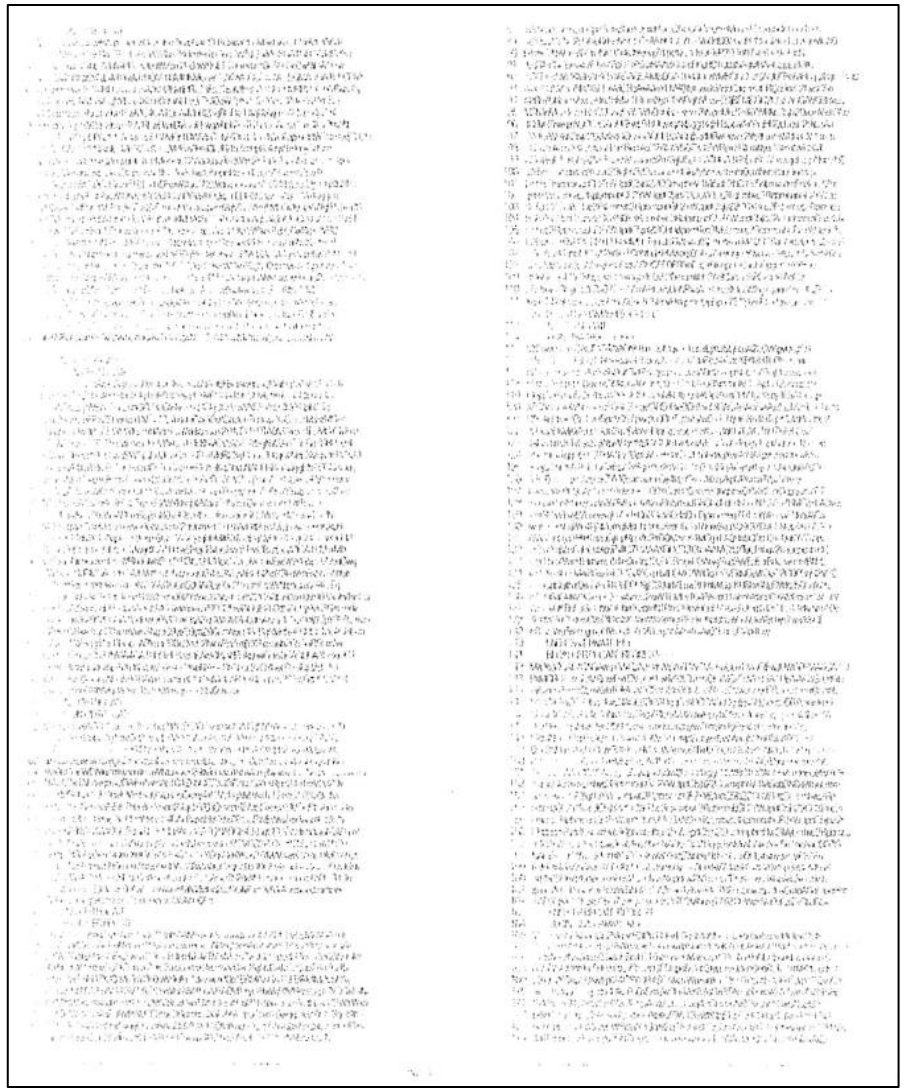


Live Russia-Ukraine wa
live: 'Heaviest of battles
ahead in Kherson, says

Lavabit and KEMs



Lavabit



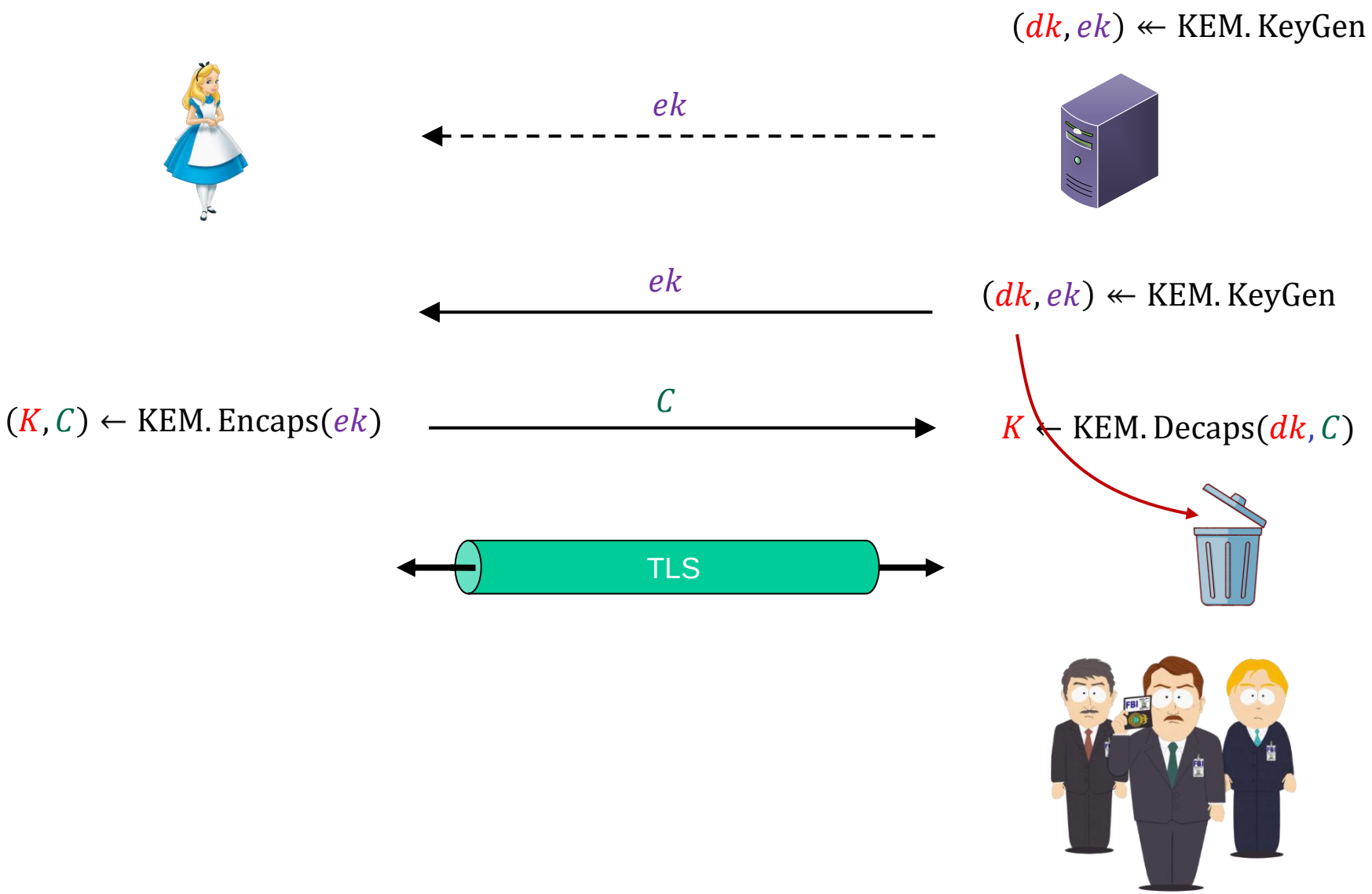
Forward secrecy

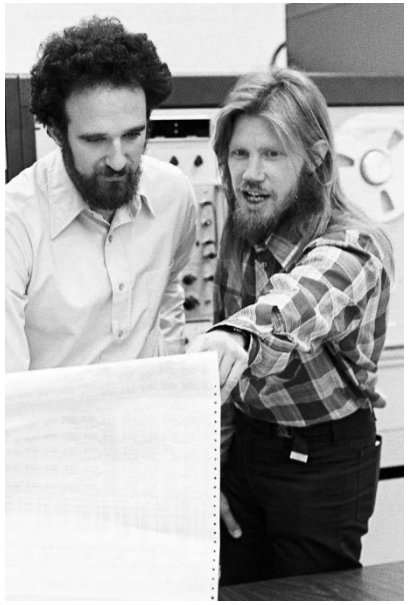
- What Lavabit's solution lacked was **forward secrecy:**

> **Definition:** A system has **forward secrecy** if an attacker cannot obtain previously sent data, even if it obtains the parties' long-term keys

- But how is this possible?

KEMs with forward secrecy







Attacking Diffie-Hellman

Diffie-Hellman – man-in-the-middle attack

$G = \langle g \rangle$



$A = g^a$

$B = g^b$



$a \leftarrow \{1, \dots, |G|\}$

$b \leftarrow \{1, \dots, |G|\}$

$Z \leftarrow B^a = g^{ab}$



$Z \leftarrow A^b = g^{ab}$

Diffie-Hellman – man-in-the-middle attack

$G = \langle g \rangle$



$A = g^a$

$B = g^b$

$X = g^x$



$a \leftarrow \{1, \ldots, |G|\}$

$b \leftarrow \{1, \ldots, |G|\}$

$Z \leftarrow B^a = g^{ab}$

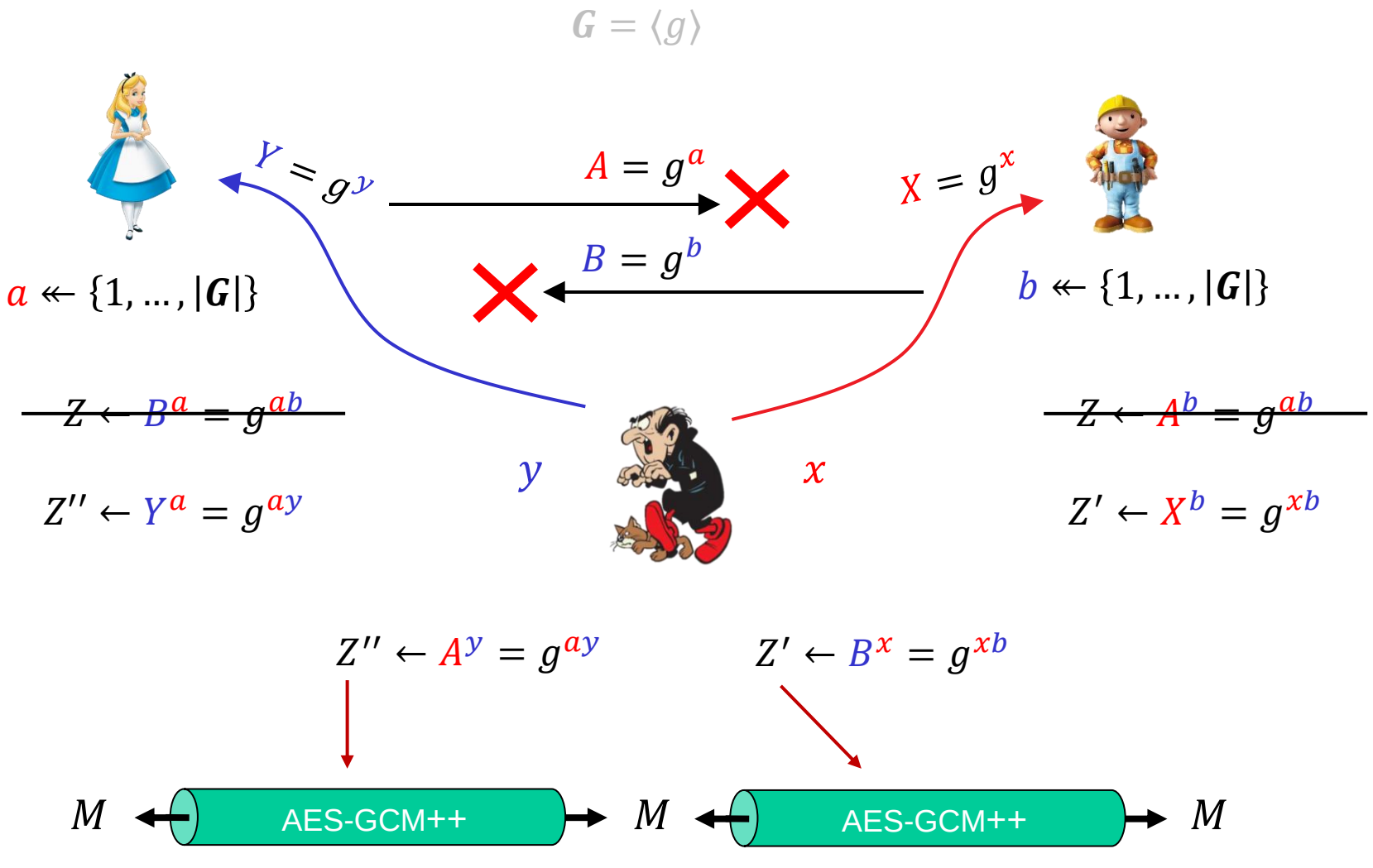
~~$Z \leftarrow A^b = g^{ab}$~~



x

$Z' \leftarrow X^b = g^{xb}$

Diffie-Hellman – man-in-the-middle attack



Noise-protocol

Long-term key

$A = g^a$

$G = \langle g \rangle, A, B$

$B = g^b$

Long-term key



$X = g^x$

$Y = g^y$



$x \leftarrow \{1, \ldots, |\boldsymbol{G}|\}$

$y \leftarrow \{1, \ldots, |\boldsymbol{G}|\}$

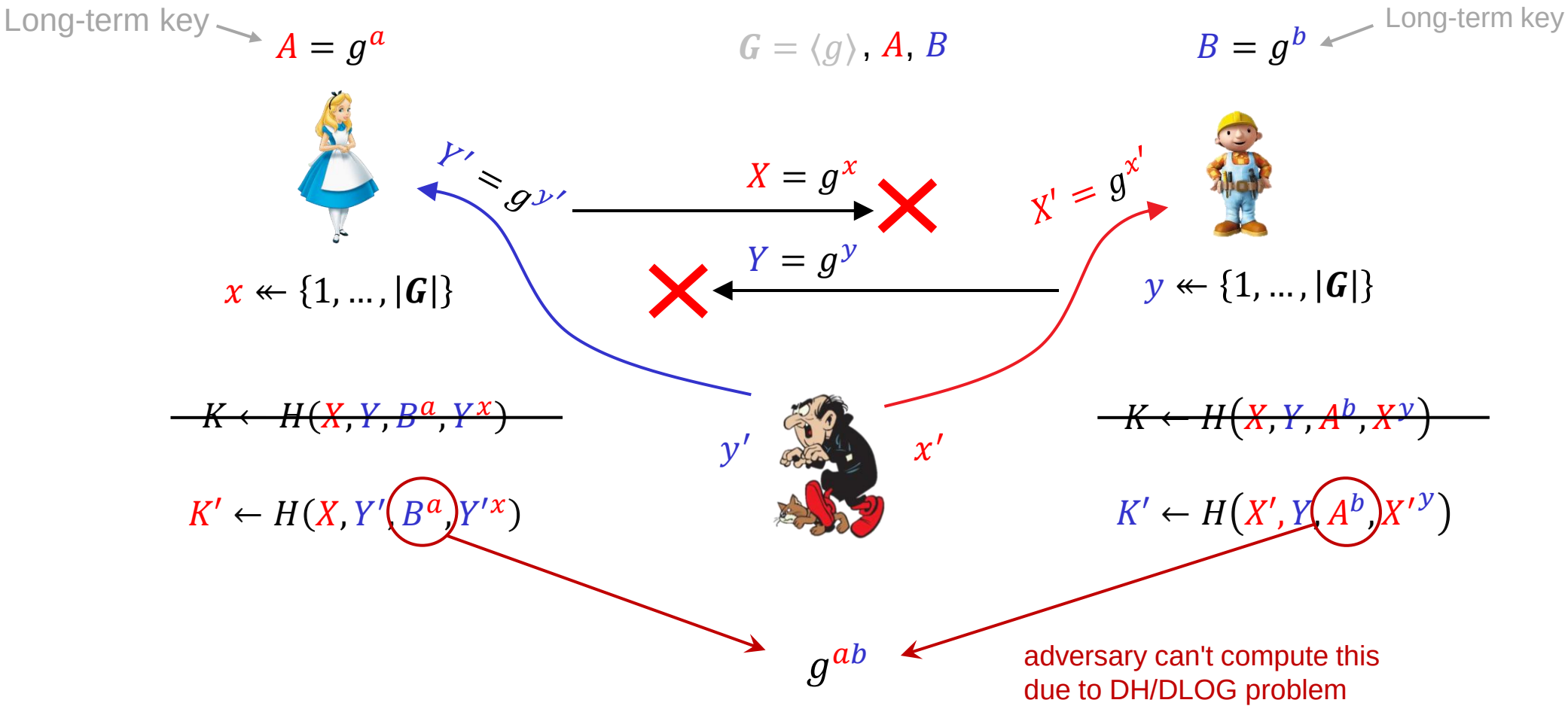
$K \leftarrow H(X, Y, B^a, Y^x)$

g^{ab} g^{xy}

$K \leftarrow H(X, Y, A^b, X^y)$

g^{ab} g^{xy}

Noise-protocol



Can't compute K' or K'!

Noise-protocol

Long-term key (static)

$A = g^a$



$G = \langle g \rangle, A, B$

$X = g^x$

$Y = g^y$

ephemeral

$B = g^b$

Long-term key (static)



$x \leftarrow \{1, \ldots, |G|\}$

$y \leftarrow \{1, \ldots, |G|\}$

Many alternatives:

$K \leftarrow H(X, Y, g^{ab}, g^{xy})$ static-static, ephemeral-ephemeral

$K \leftarrow H(X, Y, g^{ay}, g^{xb})$ static-ephemeral, ephemeral-static

$K \leftarrow H(X, Y, g^{ay}, g^{xb}, g^{xy})$ static-ephemeral, ephemeral-static, ephemeral-ephemeral

$K \leftarrow H(X, Y, g^{ab}, g^{ay}, g^{xb}, g^{xy})$ static-static, static-ephemeral, ephemeral-static, ephemeral-ephemeral

Used inside





Forward secrecy

Long-term key (static)

$A = g^a$



$G = \langle g \rangle, A, B$

$B = g^b$

Long-term key (static)



$X = g^x$

$Y = g^y$

ephemeral

$x \twoheadleftarrow \{1, \ldots, |\boldsymbol{G}|\}$

$y \twoheadleftarrow \{1, \ldots, |\boldsymbol{G}|\}$

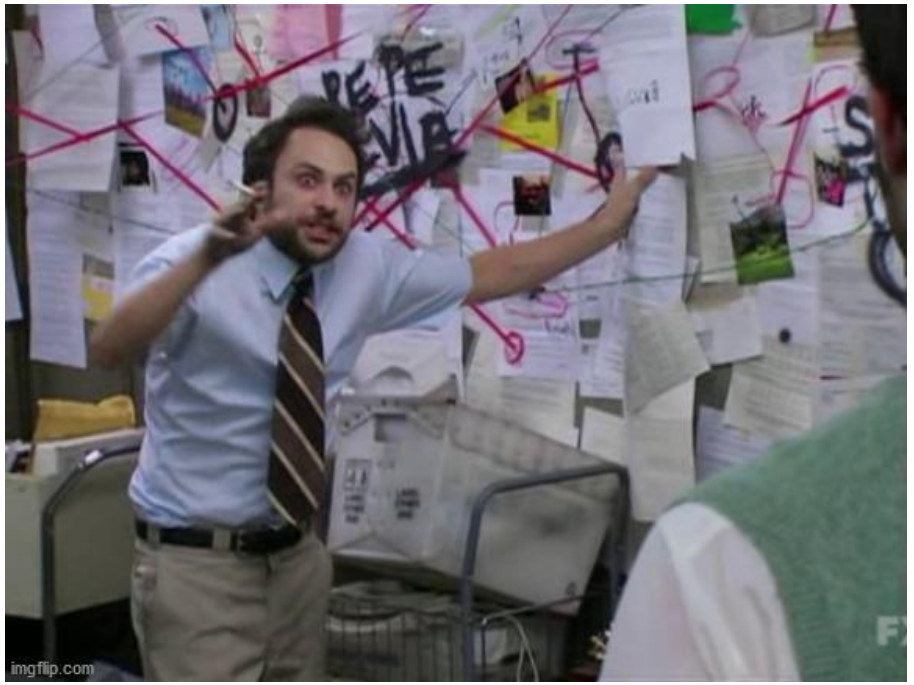
$K \leftarrow H(X, Y, g^{ab}, g^{xy})$

$K \leftarrow H(X, Y, g^{ab}, g^{xy})$

AES-GCM++

Can't compute!
(Need to solve DH or DLOG problem)





Conspiracies…

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
```

PRNG standardization

NIST Special Publication 800-90

Recommendation for Random Number Generation Using Deterministic Random Bit Generators

Elaine Barker
John Kelsey

Computer Security Division
Inforamtion 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

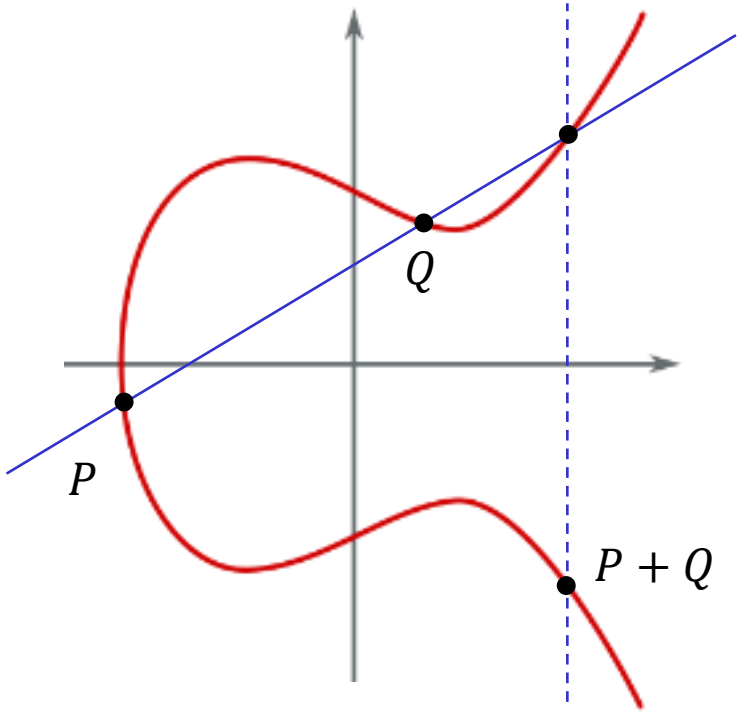
8.7.2 Additional Input 21
8.8 Prediction Resistance and Backtracking Resistance 21
9 DRBG Mechanism Functions 24
9.1 Instantiating a DRBG 24
9.2 Reseeding a DRBG Instantiation 27
9.3 Generating Pseudorandom Bits Using a DRBG 29
9.3.1 The Generate Function 29
9.3.2 Reseeding at the End of the Seedlife 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

Elliptic curves summary

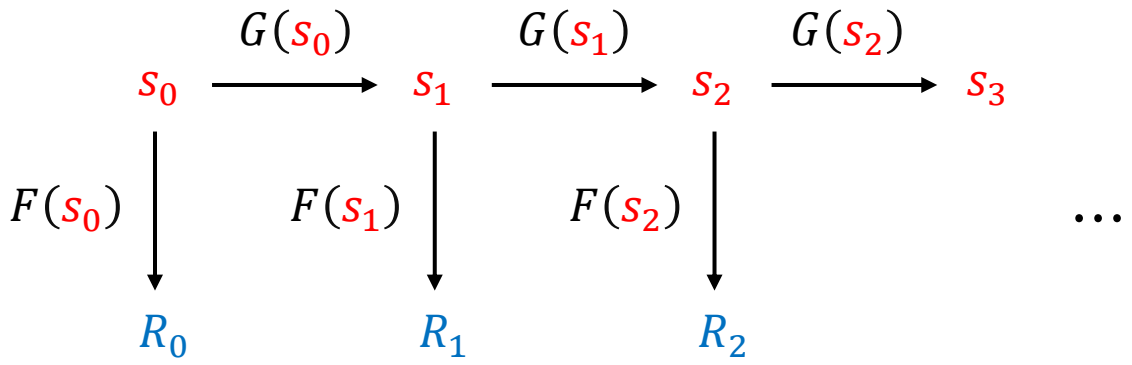
- $P = (x, y)$ point on elliptic curve
- x and y are elements in $\boldsymbol{Z}_p$
- Points P and Q can be added to get another point $P + Q$
- Special case: add P to itself n times

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

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



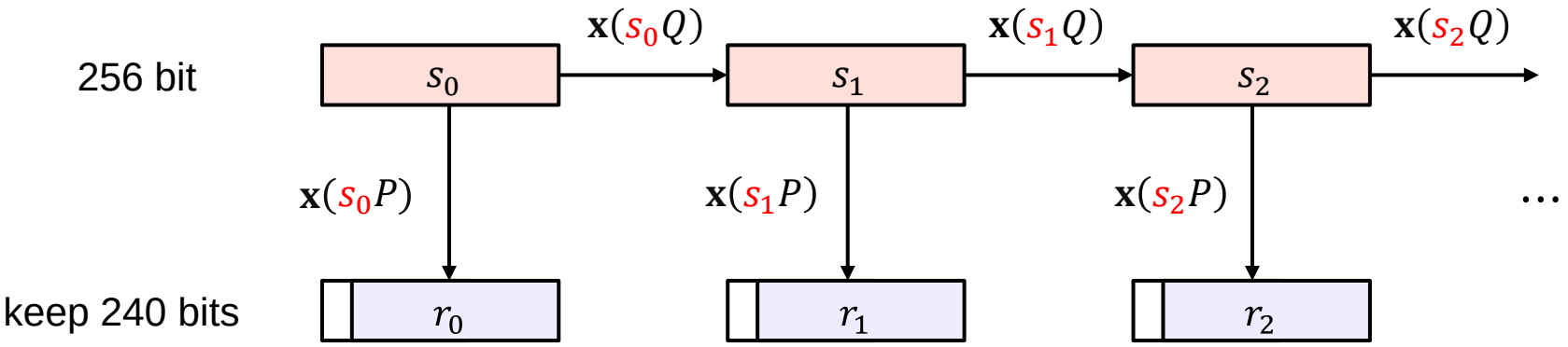
PRNG



Dual EC DRBG

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

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

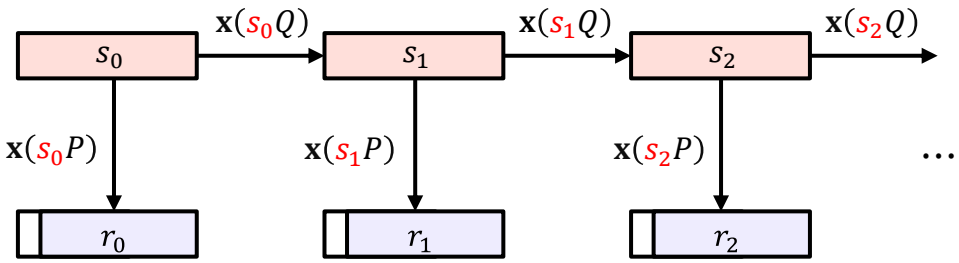


Dual EC DRBG – something's fishy

- 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 $\mathbf{Adv}^{\mathrm{prg}}_{\mathrm{DualEC}}(KG) \approx 0.0011$
  - 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?

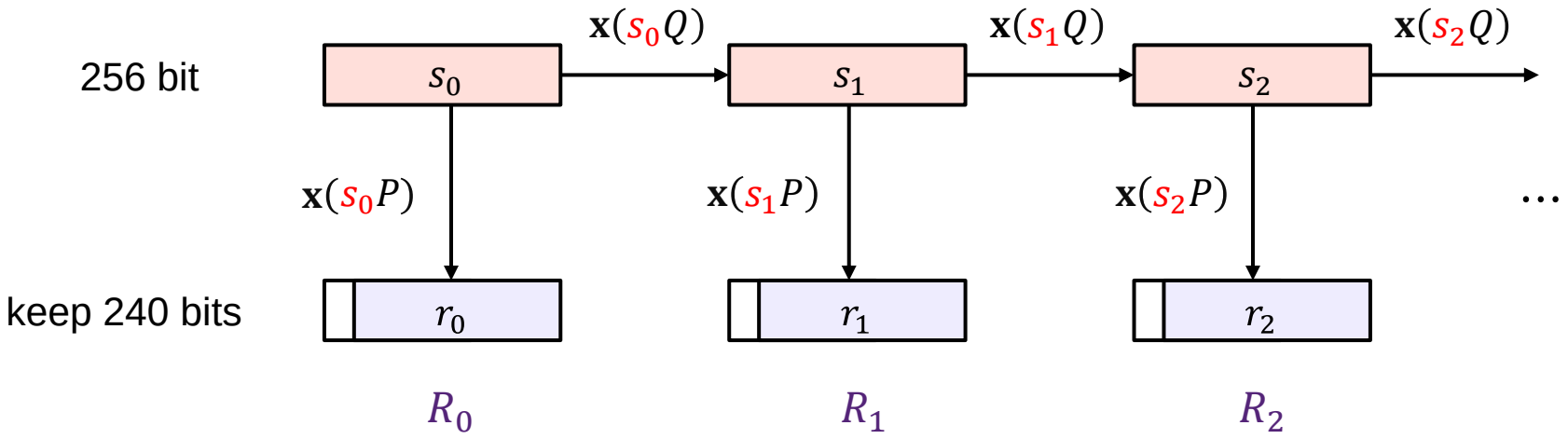


Dual EC DRBG

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

$$Q = kP$$

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

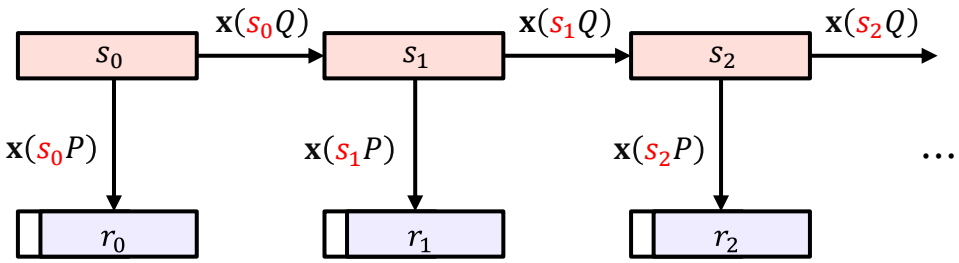


$$kR_0 = k \cdot (s_0P) = s_0 \cdot (kP) = s_0Q \xrightarrow{\mathbf{x}(\cdot)} s_1$$

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 $\mathbf{Adv}^{\text{prg}}_{\text{DualEC}}(KG) \approx 0.0011$
  - 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 $k(s_0P) = s_0(kP) = s_0Q \xrightarrow{\mathbf{x}(\cdot)} s_1$
  - 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

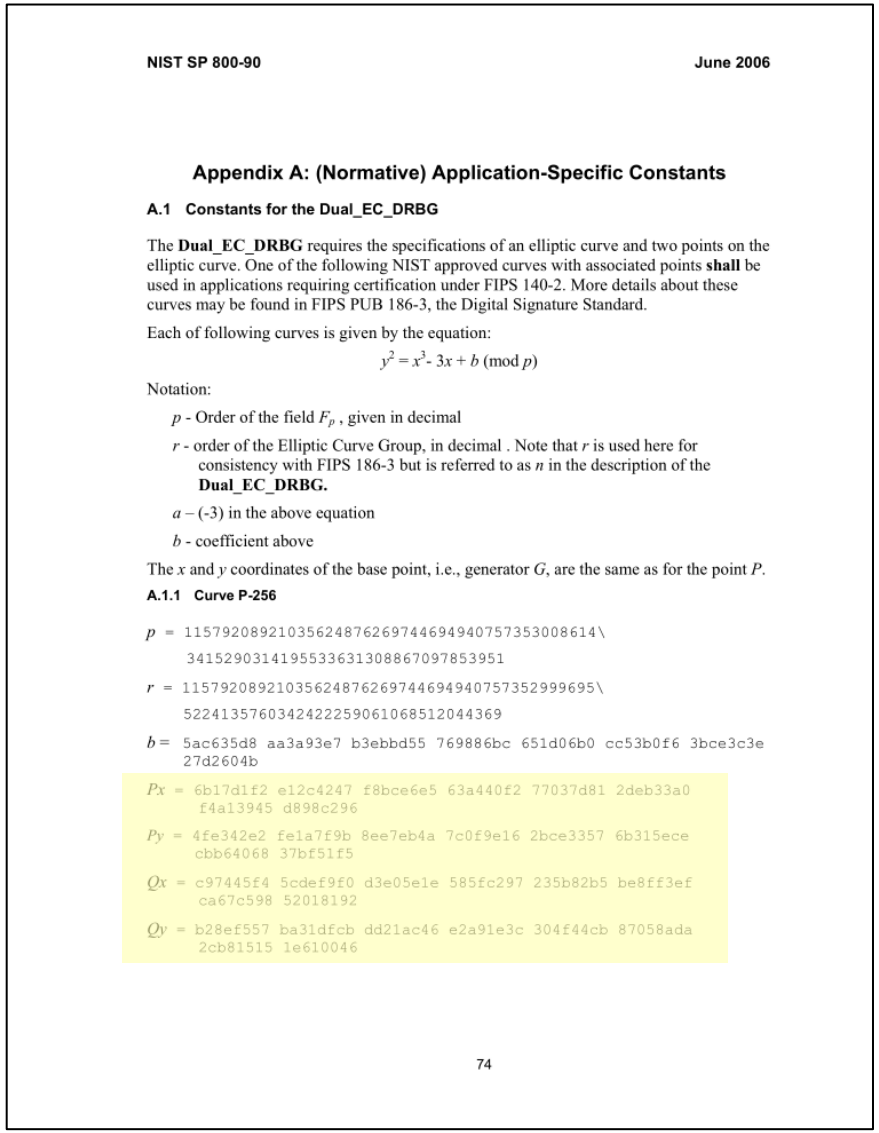
Subject: RE: Minding our Ps and Qs in Dual_EC
From: "Don Johnson" <DJohnson@cygnacom.com>
Date: Wed, October 27, 2004 11:42 am
To: "John Kelsey" <john.kelsey@nist.gov>
John,

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 = 115792089210356248762697446949407573530086143415290314195533631308867097853951

r = 115792089210356248762697446949407573529996955224135760342422259061068512044369

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

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

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

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

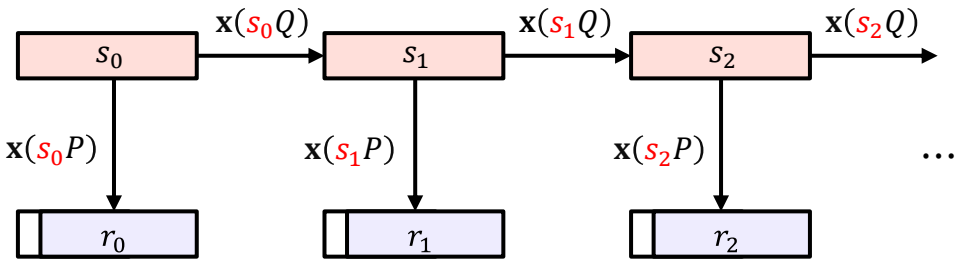
Qy = b28ef557 ba31dfcb dd21ac46 e2a91e3c 304f44cb 87058ada 2cb81515 1e610046

74

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 $\mathbf{Adv}^{\text{prg}}_{\text{DualEC}}(KG) \approx 0.0011$
  - 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 $k(s_0P) = s_0(kP) = s_0Q \xrightarrow{\mathbf{x}(\cdot)} s_1 P$
  - 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_DBRG 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], key[24], block[8];          // X9.31 vars
2.  int calls_since_reseed;
3.
4.  void prng_reseed(void) {
5.     int index = 0;                          <- local variable
6.     char tmp_output[32];                    <- local buffer
7.     system_rng_generate(tmp_output, 32);
8.     memcpy(seed, tmp_output, 8);
9.     index = 8;
10.    memcpy(key, &tmp_output[index], 24)
11.    index = 32;
12.    calls_since_reseed = 0;
13. }
14.                    caller-supplied buffer
15. void prng_generate(char *output ) {
16.    int index = 0;                          <- local variable
17.    calls_since_reseed++;
18.
19.    if (calls_since_reseed > 10 000)
20.      prng_reseed();                        <- only called conditionally
21.
22.    for (; index < 20; index += 8) {
23.      ...
24.      x9_31_generate_block(seed, key, block);
25.      ...
26.      memcpy(&output[index], block, 8);
27.    }
28. }
                    Dual EC output masked by this
```

ScreenOS v.6.2

```
1.  char seed[8], key[24], block[8];          // X9.31 vars
2.  int calls_since_reseed, index;            global variable
3.  char output[32];                          <- global shared buffer
4.
5.  void prng_reseed(void) {
6.     index = 0;
7.
8.     dual_ec_generate(output, 32);
9.     memcpy(seed, output, 8);
10.    index = 8;
11.    memcpy(key, &output[index], 24);
12.    index = 32;
13. }
14.
15. void prng_generate(void) {
16.    index = 0;
17.
18.
19.
20.    prng_reseed();                         <- called unconditionally
21.
22.    for (; index < 32; index += 8) {
23.      ...
24.      x9_31_generate_block(seed, key, block);
25.      ...
26.      memcpy(&output[index], block, 8);
27.    }
28. }
                                  …means this is skipped!
```

Source: *Where did I leave my keys? Lessons from the Juniper Dual EC incident* https://dl.acm.org/doi/10.1145/3266291

IKEv2 / IPsec

Knowledge of any of these allows deriving traffic encryption key



client nonce, xG

server nonce, yG

client authentication

server authentication



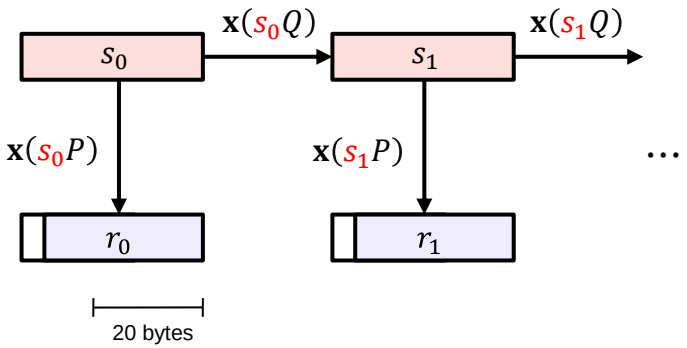


Dual EC

Juniper v6.1: 20 bytes

Juniper v6.2: 32 bytes

Created by Juniper PRNG



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 backdoor

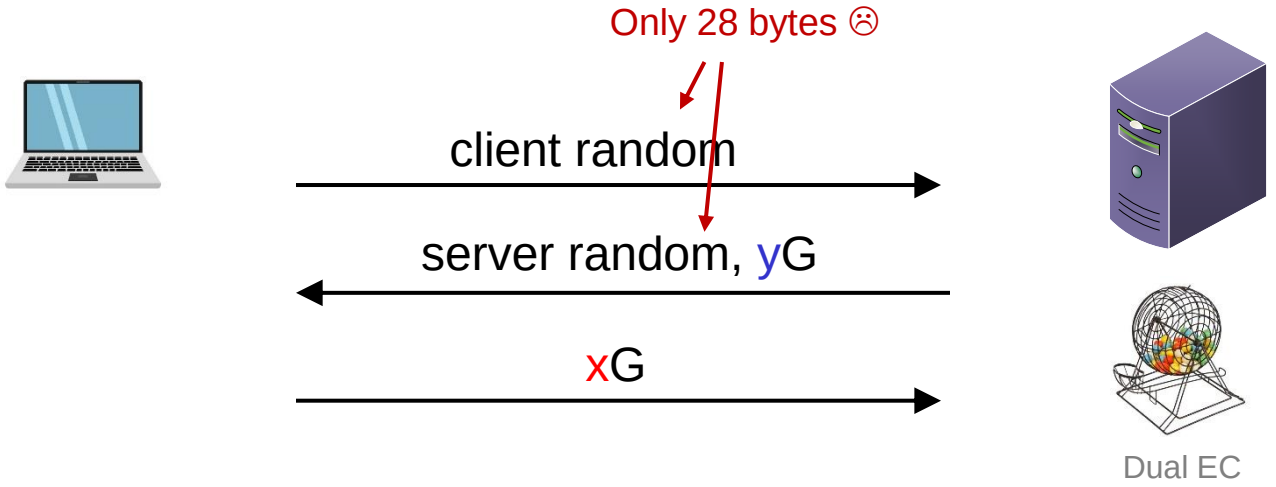
- **2008** – Juniper starts using Dual EC in ScreenOS

- **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

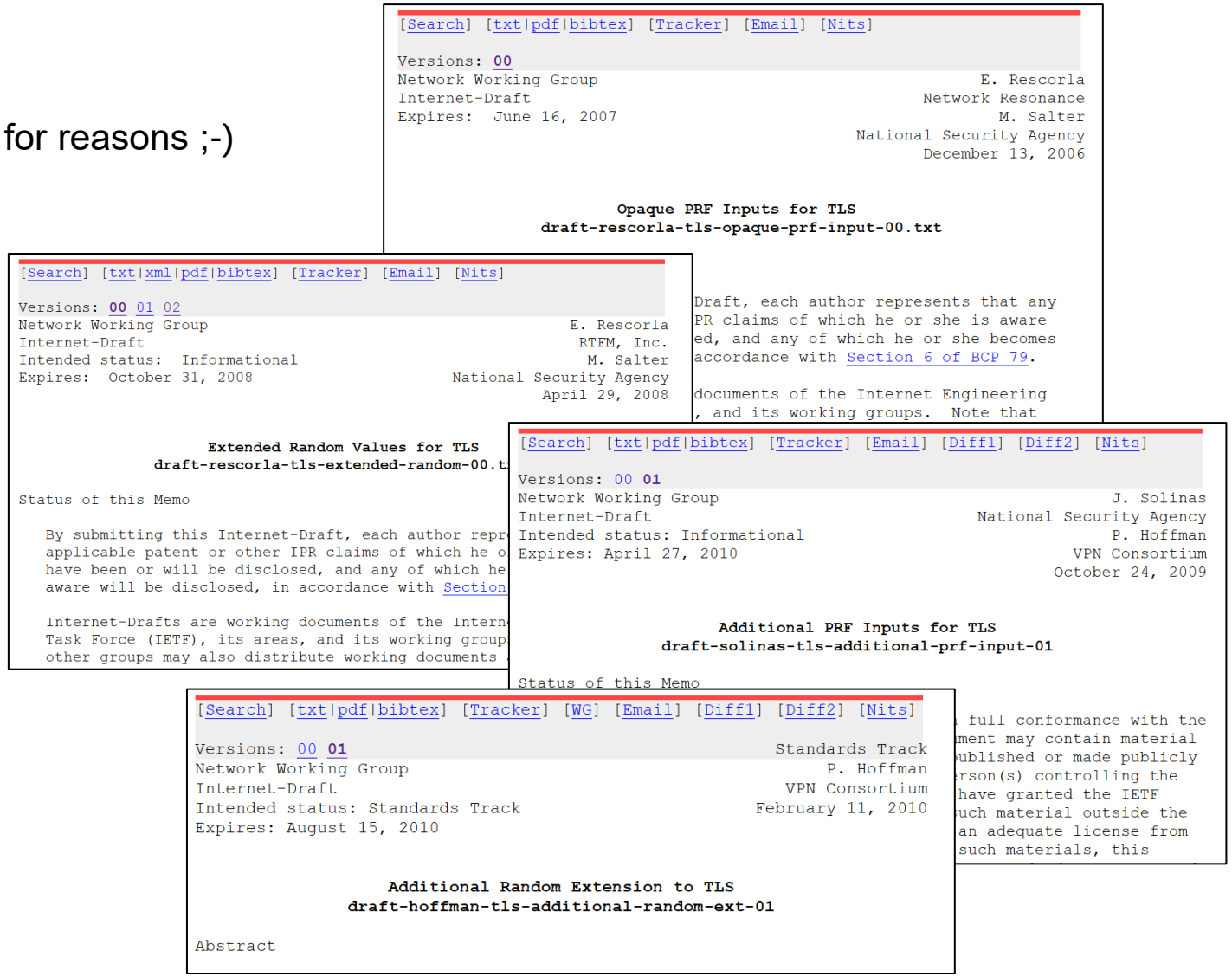


TLS 1.2



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 — E. Rescorla
Internet-Draft — Network Resonance
Expires: June 16, 2007 — M. Salter
National Security Agency
December 13, 2006

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

Draft, each author represents that any
PR claims of which he or she is aware
ed, and any of which he or she becomes
accordance with Section 6 of BCP 79.

documents of the Internet Engineering
, and its working groups. Note that

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

Versions: 00 01 02

Network Working Group — E. Rescorla
Internet-Draft — RTFM, Inc.
Intended status: Informational — M. Salter
Expires: October 31, 2008 — National Security Agency
April 29, 2008

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

Status of this Memo

By submitting this Internet-Draft, each author repr
applicable patent or other IPR claims of which he o
have been or will be disclosed, and any of which he
aware will be disclosed, in accordance with Section

Internet-Drafts are working documents of the Intern
Task Force (IETF), its areas, and its working group
other groups may also distribute working documents

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

Versions: 00 01

Network Working Group — J. Solinas
Internet-Draft — National Security Agency
Intended status: Informational — P. Hoffman
Expires: April 27, 2010 — VPN Consortium
October 24, 2009

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

Status of this Memo

full conformance with the
ment may contain material
ublished or made publicly
rson(s) controlling the
have granted the IETF
uch material outside the
an adequate license from
such materials, this

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

Versions: 00 01

Standards Track
Network Working Group — P. Hoffman
Internet-Draft — VPN Consortium
Intended status: Standards Track — February 11, 2010
Expires: August 15, 2010

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

Abstract

RSA Security – BSAFE

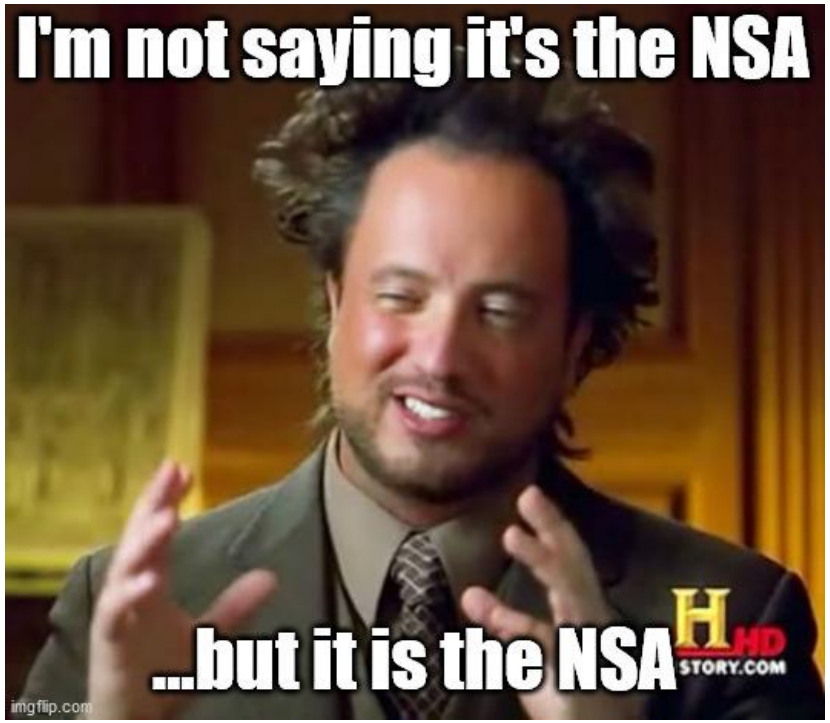
- Turns out Juniper weren't the only ones using Dual EC



- 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


I'm not saying it's the NSA
...but it is the NSA
HISTORY.COM
imgflip.com

NIST elliptic curves

- NIST standardized a number of elliptic curves (created by the NSA) in the 90's:
  - Most widely used: P-224, **P-256**, P-384, P-521

$$E: \quad y^2 = x^3 + ax + b \qquad \text{for } x, y, a, b \in \boldsymbol{Z}_p$$

Parameter	Value	Justification
p_{256}	`0xffffffff 00000001 00000000 00000000 00000000 ffffffff ffffffff ffffffff`	Efficiency ($p_{256} = 2^{256} - 2^{224} + 2^{192} - 2^{32} - 1$ allows for fast modular reduction)
$\lvert E(\boldsymbol{Z}_{p_{256}}) \rvert$	`0xffffffff 00000000 ffffffff ffffffff bce6faad a7179e84 f3b9cac2 fc632551`	Prime
a	-3	Efficiency
b	`0x5ac635d8 aa3a93e7 b3ebbd55 769886bc 651d06b0 cc53b0f6 3bce3c3e 27d2604b`	?
Generator Q	`Qx = 0x6b17d1f2 e12c4247 f8bce6e5 63a440f2 77037d81 2deb33a0 f4a13945 d898c296`	Not important (any generator will do, since the curve order is prime)

How was b generated? Chosen as the solution to $b^2 c = -27 \pmod p$

How was c generated? Defined as $c \leftarrow \mathrm{SHA1}(seed)$ for some seed $seed$

How was $seed$ generated? ~~No one knows (not even the NSA themselves apparently)!~~

$X_{i+1} \leftarrow 25214903917 \cdot X_i + 11 \pmod{2^{31}}$

Claude AI found the answer a few days ago…$seed$ was basically `lrand48(current_time)`

Next week

- How to ensure keys are authentic?
- Digital signatures
- PKI