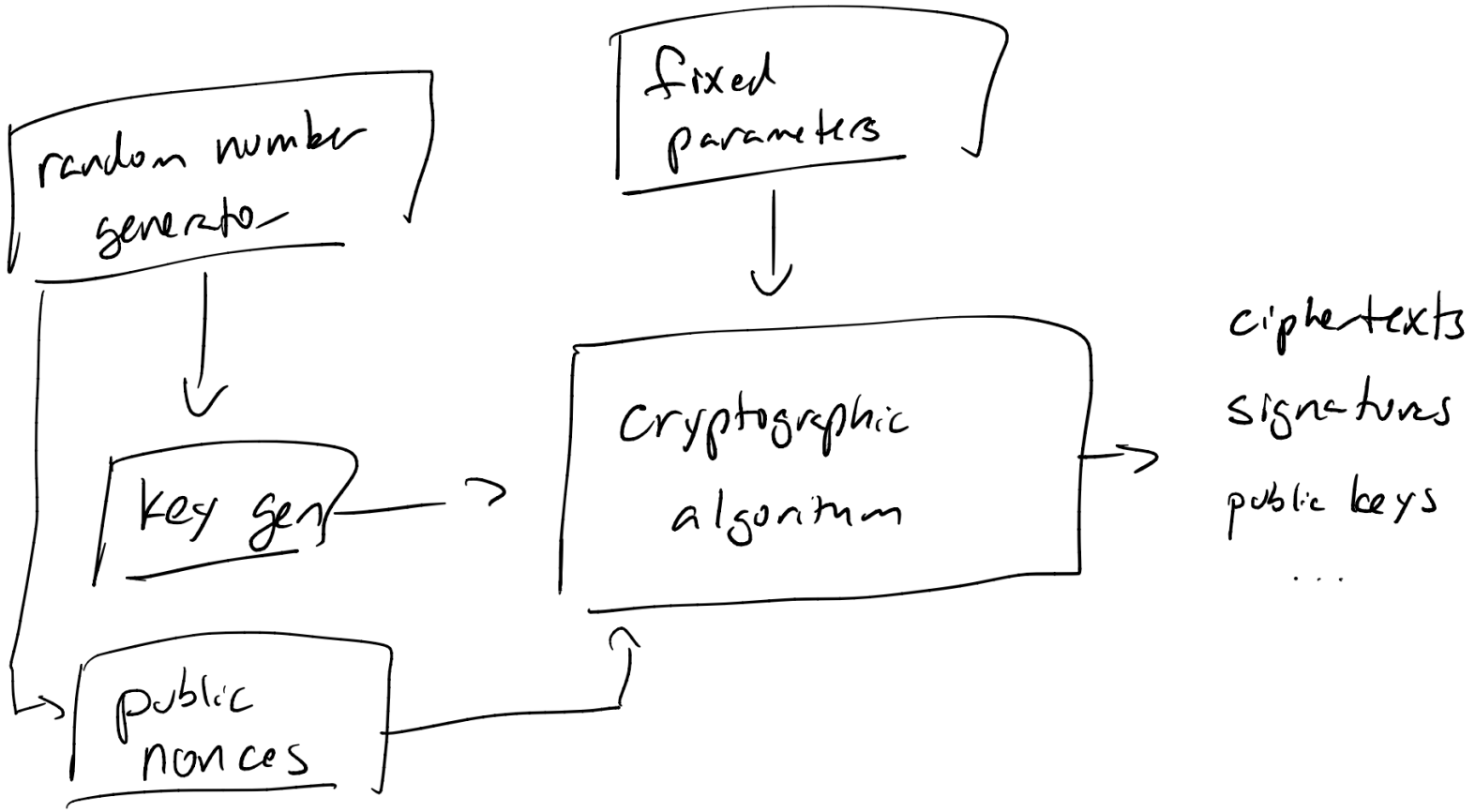
Special Request: A History of Cryptographic Backdoors

Nadia Heninger

UCSD

Cryptographic Algorithm Components



If you wanted to subvert a cryptographic algorithm, how would you do it?

Suggested methods to subvert cryptography

- Design algorithm so that true key strength is less than apparent key strength.
- Choose “fixed” parameters to weaken algorithm strength.
- Choose “fixed” parameters to encode a secret.
- Weaken key generation algorithm to generate keys with less entropy.
- Use a flawed random number generator so that secrets are easier to predict.
- …

Black Chamber: Forerunner of the NSA

Founded post WWI.

Closed down in 1929.

Henry L. Stimson:
“Gentlemen do not read each other’s mail.”

Crypto AG

Swiss company founded after WWII by Boris Hagelin.

1950s–1960s: Company paid by CIA to weaken algorithms.

1970: Bought in secret by CIA and German BND.

Machines used by dozens of countries from 1950s to 2000s.

Employees: “The algorithms always looked fishy.” “Not all questions appeared to be welcome.”

WaPo: “the secret partners adopted a set of principles for rigged algorithms… They had to be ‘undetectable by usual statistical tests’ and, if discovered, be ‘easily masked as implementation or human errors.’”

Decades of rumors confirmed in 2019.

`https://www.washingtonpost.com/graphics/2020/world/national-security/cia-crypto-encryption-machines-espionage/`

Late 1970s: DES

NSA made two changes to IBM's algorithm:

- Changed key strength from 64 to 56 bits: overt weakening.

- Changed S-boxes. Suspected to be a backdoor but later discovered to protect against differential cryptanalysis.

The “crypto wars” in the US

- Crypto wars 1.0
  - Late 1970s,
  - US government threatened legal sanctions on researchers who published papers about cryptography.
  - Threats to retroactively classify cryptography research.
- Crypto wars 2.0
  - 1990s
  - Main isssues: Export control and key escrow
  - Several legal challenges
- Crypto wars 3.0
  - Now
  - Snowden
  - Apple v. FBI
  - …?
  - Calls for “balance”

US export controls on cryptography

- Pre-1994: Encryption software requires individual export license as a munition.
- 1994: US State Department amends ITAR regulations to allow export of approved software to approved countries without individual licenses. 40-bit symmetric cryptography was understood to be approved.
- 1995: Netscape develops initial SSL protocol. Includes weakened “export” cipher suites.
- 1996: Bernstein v. United States; California judge rules ITAR regulations are unconstitutional because “code is speech”
- 1996: Cryptography regulation moved to Department of Commerce.
- 1999: TLS 1.0 standardized. Includes weakened “export” cipher suites.
- 2000: Department of Commerce loosens regulations on mass-market and open source software.

International Traffic in Arms Regulations

April 1, 1992 version

Category XIII--Auxiliary Military Equipment ...

(b) Information Security Systems and equipment, cryptographic devices, software, and components specifically designed or modified therefore, including:

(1) Cryptographic (including key management) systems, equipment, assemblies, modules, integrated circuits, components or software with the capability of maintaining secrecy or confidentiality of information or information systems, except cryptographic equipment and software as follows:

(i) Restricted to decryption functions specifically designed to allow the execution of copy protected software, provided the decryption functions are not user-accessible.

(ii) Specially designed, developed or modified for use in machines for banking or money transactions, and restricted to use only in such transactions. Machines for banking or money transactions include automatic teller machines, self-service statement printers, point of sale terminals or equipment for the encryption of interbanking transactions.

...

Commerce Control List, March 2021

2.a.A ‘‘symmetric algorithm’’ employing a key length in excess of 56 bits, not including parity bits;

2.b.An ‘‘asymmetric algorithm’’ where the security of the algorithm is based on any of the following:

2.b.1. Factorization of integers in excess of 512 bits (e.g., RSA);

2.b.2. Computation of discrete logarithms in a multiplicative group of a finite field of size greater than 512 bits (e.g., Diffie-Hellman over Z/pZ); or

2.b.3. Discrete logarithms in a group other than mentioned in paragraph 2.b.2 of this Technical Note in excess of 112 bits (e.g., Diffie-Hellman over an elliptic curve); or

2.c. An ‘‘asymmetric algorithm’’ where the security of the algorithm is based on any of the following:

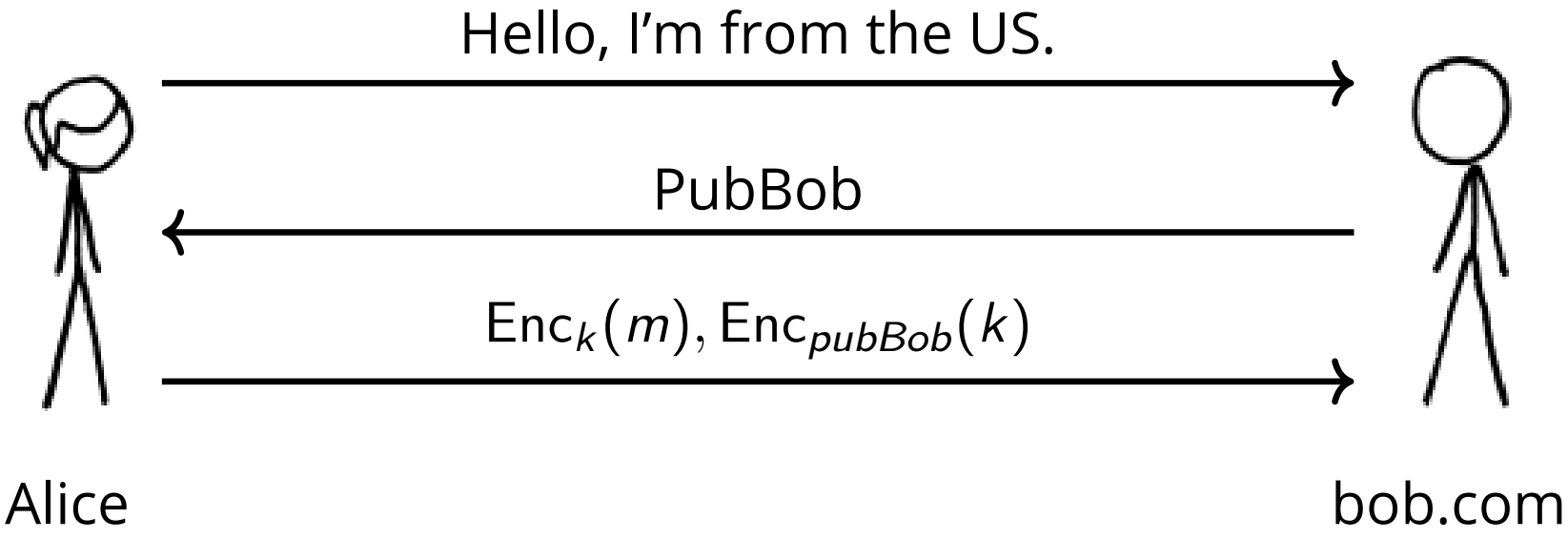
2.c.1. Shortest vector or closest vector problems associated with lattices (e.g., NewHope, Frodo, NTRUEncrypt, Kyber, Titanium);

2.c.2. Finding isogenies between Supersingular elliptic curves (e.g., Supersingular Isogeny Key Encapsulation); or

2.c.3. Decoding random codes (e.g., McEliece, Niederreiter).

“Export” cipher strength negotiation

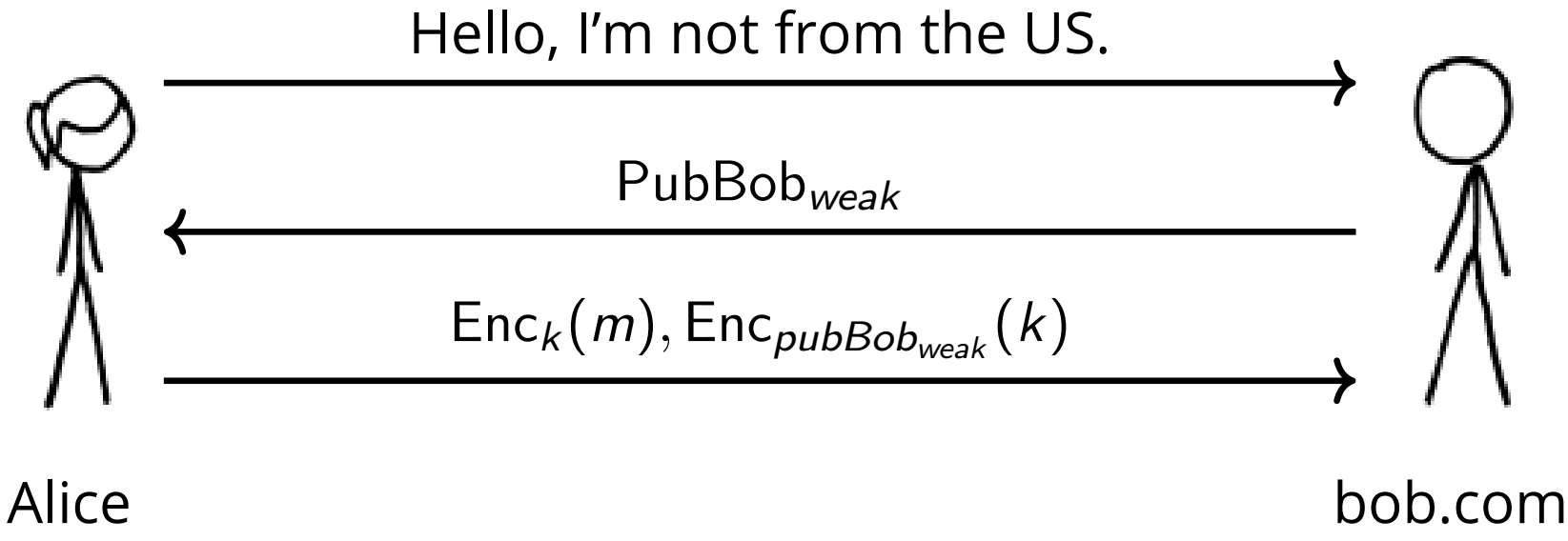
Case # 1: Non-export controlled software



PubBob is a strong public key and only Bob can decrypt the message.

“Export” cipher strength negotiation

Case # 2: Export controlled software



- PubBob is weakened so that a large government could decrypt if significant resources invested.
- However, computation is not feasible for public, so web is safe for consumers.

Multi-decade fallout from US crypto export control

- Discouraged business in US
- Support for deliberately weakened “export-grade” cipher suites did not disappear in 2000, because vendors maintained backwards compatibility.
- 2015: FREAK, LogJam, and DROWN attacks exploited previously undiscovered SSL/TLS protocol flaws around negotiating export cipher suites. **10-25%** of popular web sites vulnerable.
- First public 512-bit factorization in 1999.
  - By 2015, 512-bit RSA could be factored by anyone for $75 in 3-4 hours on cloud computing.

2G cipher weak

GEA-1 cipher designed in France in 1998 for use in 2G.

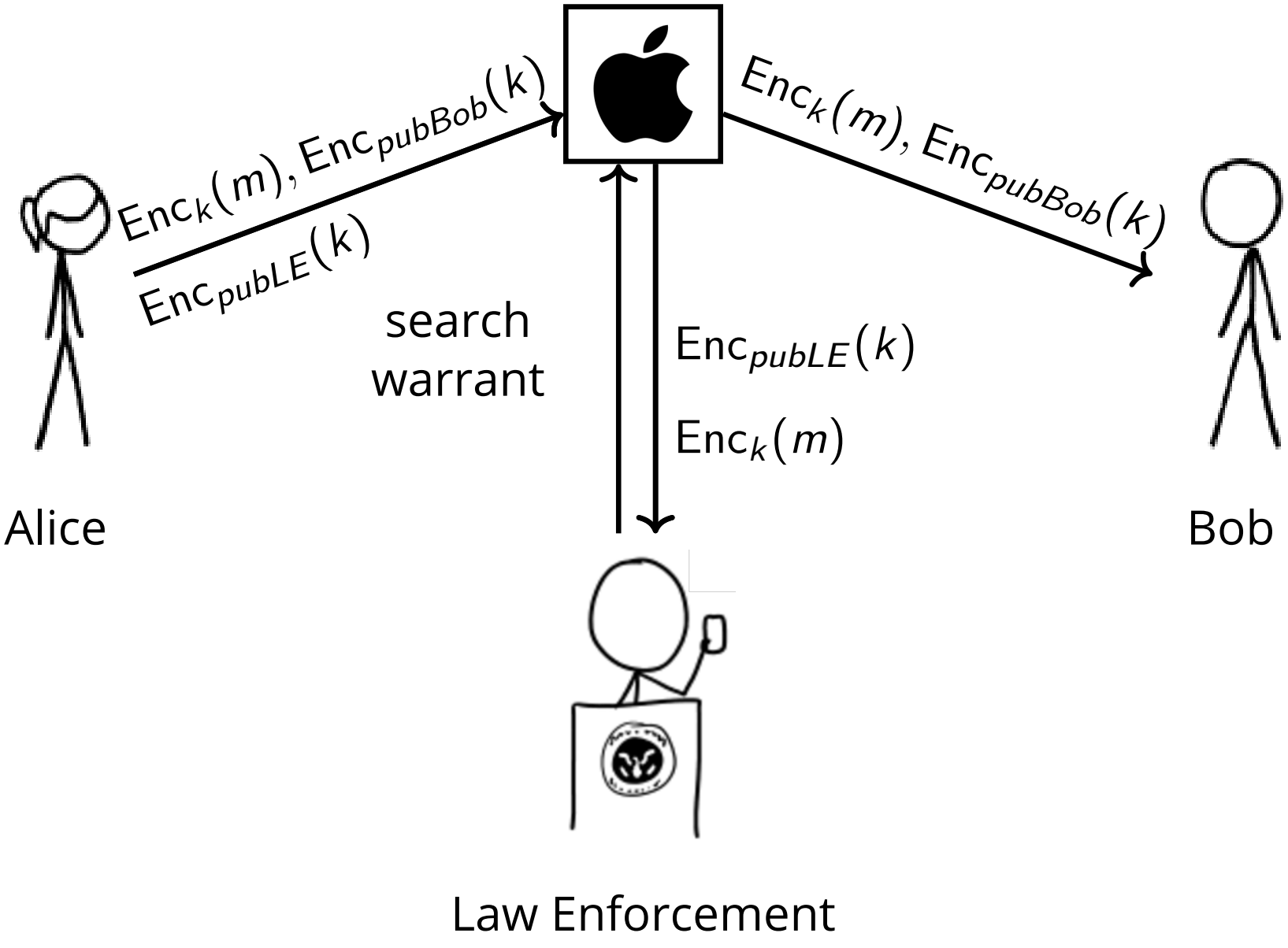
2021: Researchers discover 2^{40} attack.

“It was explicitly mentioned as a design requirement that ‘the algorithm should be generally exportable taking into account current export restrictions’ and that ‘the strength should be optimized taking into account the above requirement’

Hypothesis: Algorithm designed to offer exactly 40 bits of security to comply with European export restrictions.

`https://eprint.iacr.org/2021/819.pdf`

A basic key escrow system



1993: NSA promotes “Clipper chip” key escrow

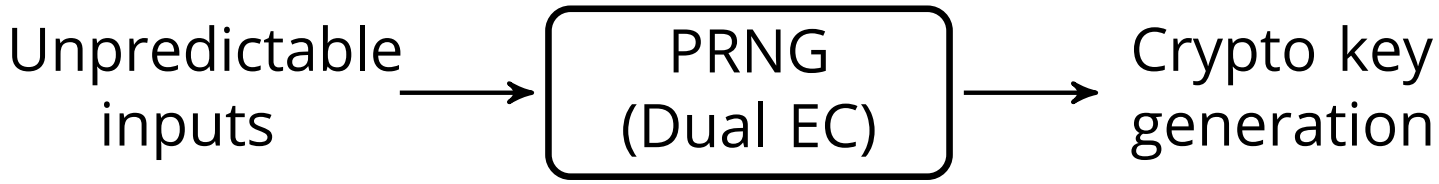
- Intended for voice transmission by telecommunications companies.
- Used Diffie-Hellman key exchange and Skipjack NSA-designed symmetric cipher with an 80-bit key.
- Secret keys were transmitted in a “Law Enforcement Access Field” to allow decryption with a warrant.
- 1994: Matt Blaze publishes protocol flaw allowing circumvention of key escrow.
- System abandoned by 1996.

Key escrow in theory and in practice

- In theory, key escrow is provably secure.
- In practice, schemes are difficult to secure.

Dual EC DRBG

- Pseudorandom number generator (PRNG) standardized by NIST, ISO.
- How to use a PRNG for cryptography:



- Dual EC design encodes backdoor/key escrow potential:
  - Algorithm designer can recover cryptographic secrets.
  - Cryptographically secure against all other parties.

Dual EC DRBG

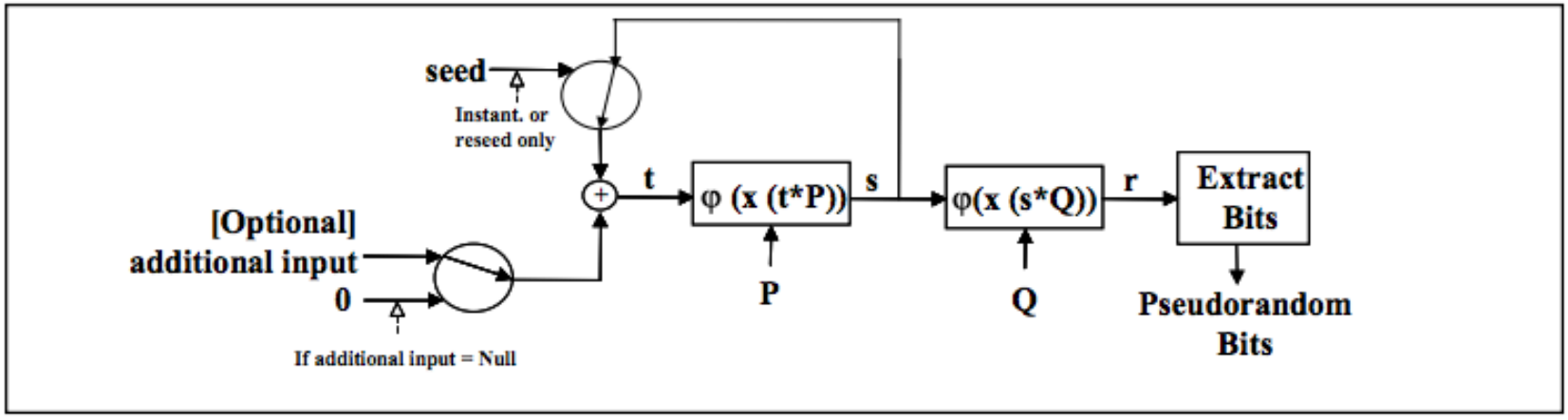


Figure 13: Dual_EC_DRBG

- Parameters: Pre-specified elliptic curve points P and Q.
- Seed: 32-byte integer s
- State: x-coordinate of point sP. ($\phi(x(sP))$ above.)
- Update: $t = s \oplus$ optional additional input. State $s = x(tP)$.
- Output: At state s, compute x-coordinate of point $x(sQ)$, discard top 2 bytes, output 30 bytes.

Timeline of Dual EC DRBG scandal

- Early 2000s: Created by the NSA and pushed towards standardization
- 2004: Published as part of ANSI X9.82 part 3 draft
- 2004: RSA makes Dual EC the default PRNG in BSAFE
- 2005: Standardized in NIST SP 800-90 draft
- 2007: Shumow and Ferguson demonstrate theoretical backdoor
- 2013: Snowden documents lead to renewed interest in Dual EC
- 2014: Practical attacks on TLS using Dual EC demonstrated
- 2015: NIST removes Dual EC from list of approved PRNGs

Still no way to prove standard was backdoored, or compromise traffic without knowing secret parameters.

How to exploit Dual EC backdoor

Shumow and Ferguson 2007

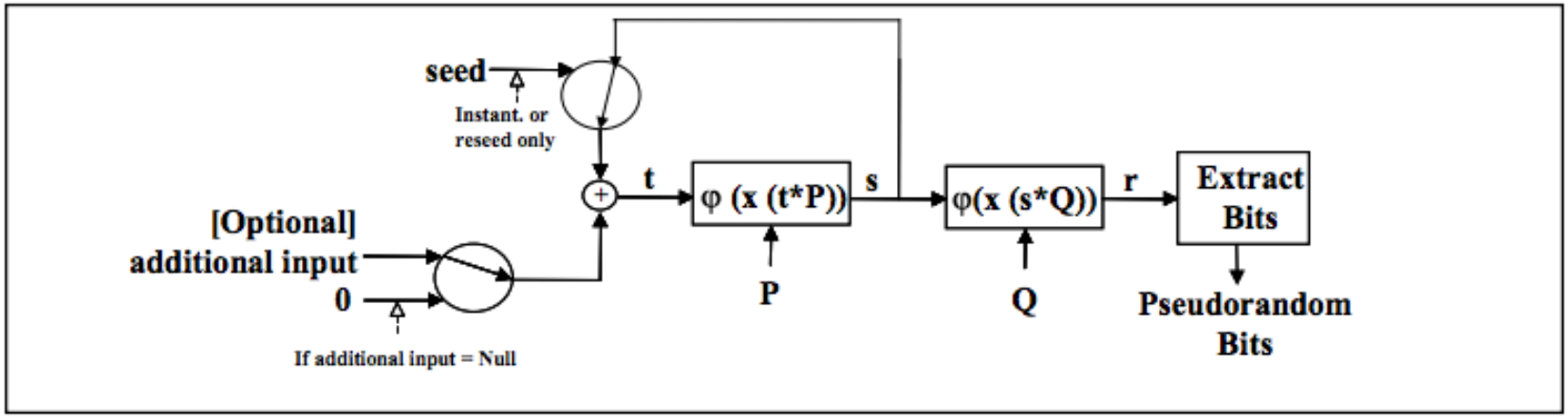


Figure 13: Dual_EC_DRBG

1. Assume attacker controls standard and constructs points with known relationship $P = dQ$.
2. Attacker gets 30 bytes of x-coordinate of sQ. Attacker brute forces 2^{16} MSBs, gets 2^{17} possible y-coordinates, ends up with 2^{15} candidates for sQ.
3. For each candidate sQ attacker computes $dsQ = sP$ and compares to next output.

September 2013: NSA Bullrun in NY Times

- (TS//SI//REL TO USA, FVEY) Insert vulnerabilities into commercial encryption systems, IT systems, networks, and endpoint communications devices used by targets.
- (TS//SI//REL TO USA, FVEY) Collect target network data and metadata via cooperative network carriers and/or increased control over core networks.
- (TS//SI//REL TO USA, FVEY) Leverage commercial capabilities to remotely deliver or receive information to and from target endpoints.
- (TS//SI//REL TO USA, FVEY) Exploit foreign trusted computing platforms and technologies.
- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.
- (TS//SI//REL TO USA, FVEY) Make specific and aggressive investments to facilitate the development of a robust exploitation capability against Next-Generation Wireless (NGW) communications.

Dual EC Attack Complexity in TLS Implementations

Checkoway et al. 2014

Table 1: Summary of our results for Dual EC using NIST P-256.

Library	Default PRNG	Cache Output	Ext. Random	Bytes per Session	Adin Entropy	Attack Complexity	Time (minutes)
BSAFE-C v1.1	✓	✓	✓†	31–60	—	$30 \cdot 2^{15}(C_v + C_f)$	0.04
BSAFE-Java v1.1	✓		✓†	28	—	$2^{31}(C_v + 5C_f)$	63.96
SChannel I‡				28	—	$2^{31}(C_v + 4C_f)$	62.97
SChannel II‡				30	—	$2^{33}(C_v + C_f) + 2^{17}(5C_f)$	182.64
OpenSSL-fixed I*				32	20	$2^{15}(C_v + 3C_f) + 2^{20}(2C_f)$	0.02
OpenSSL-fixed III**				32	$35 + k$	$2^{15}(C_v + 3C_f) + 2^{35+k}(2C_f)$	$2^k \cdot 83.32$

* Assuming process ID and counter known. ** Assuming 15 bits of entropy in process ID, maximum counter of 2^k. See Section 4.3.
† With a library–compile-time flag. ‡ Versions tested: Windows 7 64-bit Service Pack 1 and Windows Server 2010 R2.

the grugq
@thegrugq

Follow

Woah! Juniper discovers a backdoor to decrypt VPN traffic (and remote admin) has been inserted into their OS source

Important Announcement about ScreenOS®

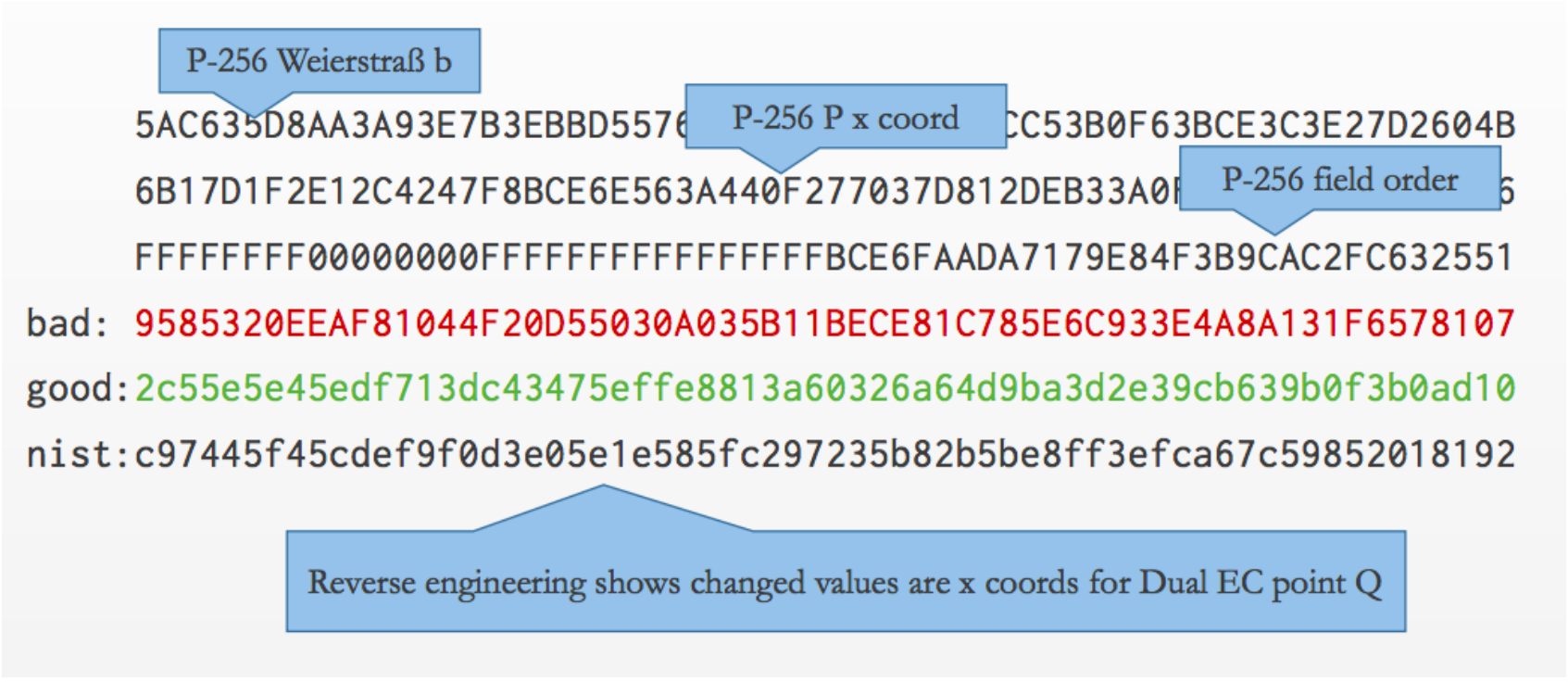
IMPORTANT JUNIPER SECURITY ANNOUNCEMENT CUSTOMER UPDATE: DECEMBER 20, 2015 Administrative Access (CVE-2015-7755) only affects ScreenOS 6.3.0r17 through

forums.juniper.net

Juniper ScreenOS Dual EC attack

- 2008: Juniper adopts Dual EC with countermeasures against theoretical backdoor.
- 2012: Unidentified attackers modified Juniper ScreenOS Dual EC implementation.
- 2015: Juniper discovers code modifications and publishes security advisory.
- 2016: Checkoway et al. observe that:
  1. Juniper's 2008 countermeasures contained a subtle bug: implementation enabled backdoor.
  2. Attacker had changed backdoor parameters.

Juniper ScreenOS code diff



Juniper cascaded Dual EC with ANSI X9.31 RNG

- ScreenOS only FIPS validated for ANSI X9.31, not Dual EC
- Juniper used non-default points for Dual EC

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

ScreenOS RNG implementation

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }
}

void prng_reseed(void) {
  blocks_generated_since_reseed = 0;
  if (dualec_generate(prng_temporary, 32) != 32)
    error_handler("FIPS ERROR: PRNG failure, unable to reseed\n", 11);
  memcpy(prng_seed, prng_temporary, 8);
  prng_output_index = 8;
  memcpy(prng_key, &prng_temporary[prng_output_index], 24);
  prng_output_index = 32;
}
```

ScreenOS RNG implementation

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();      // conditional reseed
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }
}

void prng_reseed(void) {
  blocks_generated_since_reseed = 0;
  if (dualec_generate(prng_temporary, 32) != 32)
    error_handler("FIPS ERROR: PRNG failure, unable to reseed\n", 11);
  memcpy(prng_seed, prng_temporary, 8);
  prng_output_index = 8;
  memcpy(prng_key, &prng_temporary[prng_output_index], 24);
  prng_output_index = 32;
}
```

ScreenOS RNG implementation

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }
}

void prng_reseed(void) {
  blocks_generated_since_reseed = 0;
  if (dualec_generate(prng_temporary, 32) != 32)    // generate Dual EC output
    error_handler("FIPS ERROR: PRNG failure, unable to reseed\n", 11);
  memcpy(prng_seed, prng_temporary, 8);
  prng_output_index = 8;
  memcpy(prng_key, &prng_temporary[prng_output_index], 24); // copy output
  prng_output_index = 32;
}
```

ScreenOS RNG implementation

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block); // gen output
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }
}

void prng_reseed(void) {
  blocks_generated_since_reseed = 0;
  if (dualec_generate(prng_temporary, 32) != 32)
    error_handler("FIPS ERROR: PRNG failure, unable to reseed\n", 11);
  memcpy(prng_seed, prng_temporary, 8);
  prng_output_index = 8;
  memcpy(prng_key, &prng_temporary[prng_output_index], 24);
  prng_output_index = 32;
}
```

ScreenOS RNG implementation

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;      // global variable
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())       // always true
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }
}

void prng_reseed(void) {
  blocks_generated_since_reseed = 0;
  if (dualec_generate(prng_temporary, 32) != 32)
    error_handler("FIPS ERROR: PRNG failure, unable to reseed\n", 11);
  memcpy(prng_seed, prng_temporary, 8);
  prng_output_index = 8;
  memcpy(prng_key, &prng_temporary[prng_output_index], 24);
  prng_output_index = 32;
}
```

ScreenOS RNG implementation

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }
}

void prng_reseed(void) {
  blocks_generated_since_reseed = 0;
  if (dualec_generate(prng_temporary, 32) != 32)     // global variable
    error_handler("FIPS ERROR: PRNG failure, unable to reseed\n", 11);
  memcpy(prng_seed, prng_temporary, 8);
  prng_output_index = 8;
  memcpy(prng_key, &prng_temporary[prng_output_index], 24);
  prng_output_index = 32;   // set to 32
}
```

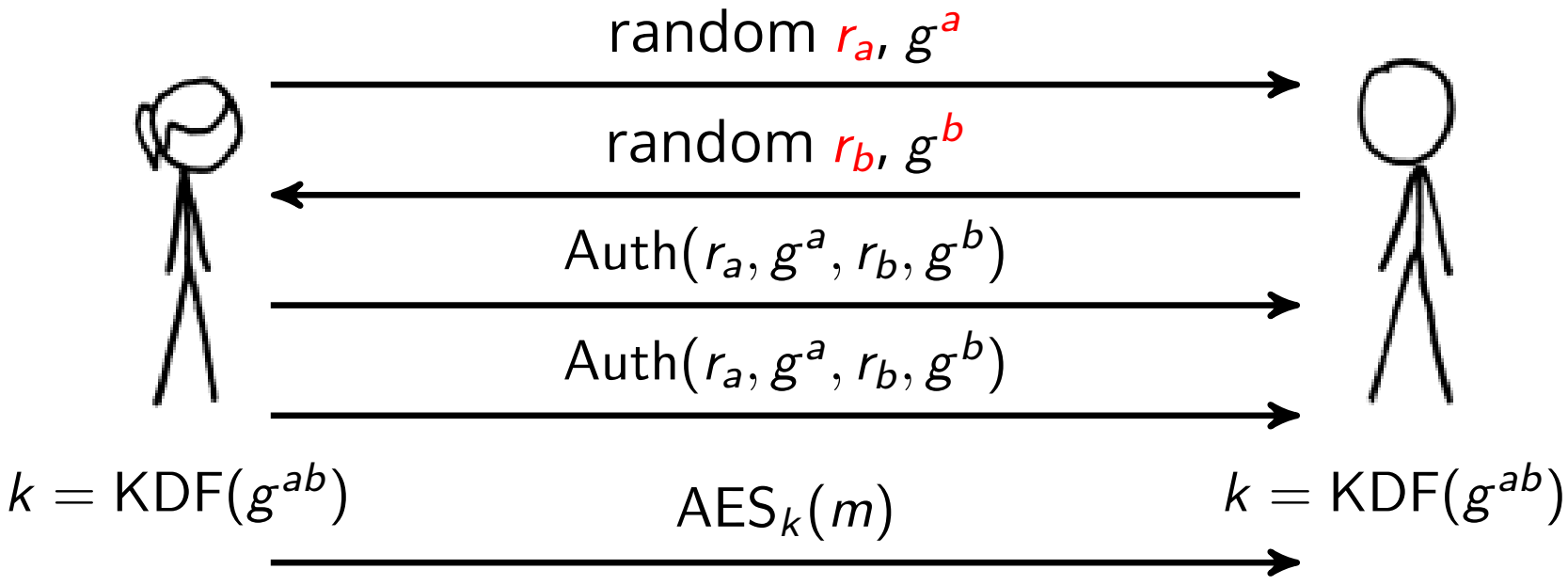
ScreenOS RNG implementation

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) { // never runs
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8); // reuses buffer
  }
}

void prng_reseed(void) {
  blocks_generated_since_reseed = 0;
  if (dualec_generate(prng_temporary, 32) != 32)
    error_handler("FIPS ERROR: PRNG failure, unable to reseed\n", 11);
  memcpy(prng_seed, prng_temporary, 8);
  prng_output_index = 8;
  memcpy(prng_key, &prng_temporary[prng_output_index], 24);
  prng_output_index = 32;
}
```

ScreenOS RNG implementation

```
void prng_generate(void) {
  int time[2];
  time[0] = 0;
  time[1] = get_cycles();
  prng_output_index = 0;
  ++blocks_generated_since_reseed;
  if (!one_stage_rng())
    prng_reseed();
  for (; prng_output_index <= 0x1F; prng_output_index += 8) {
    // FIPS checks removed for clarity
    x9_31_generate_block(time, prng_seed, prng_key, prng_block);
    // FIPS checks removed for clarity
    memcpy(&prng_temporary[prng_output_index], prng_block, 8);
  }  // output is raw Dual EC output!
}

void prng_reseed(void) {
  blocks_generated_since_reseed = 0;
  if (dualec_generate(prng_temporary, 32) != 32)
    error_handler("FIPS ERROR: PRNG failure, unable to reseed\n", 11);
  memcpy(prng_seed, prng_temporary, 8);
  prng_output_index = 8;
  memcpy(prng_key, &prng_temporary[prng_output_index], 24);
  prng_output_index = 32;
}
```

Passive state recovery in ScreenOS IPsec



- Use random nonces to carry out state recovery attack.
- ScreenOS used 32-byte nonce $\implies$ efficient attack.
- After state recovered, then recover secret exponents.
- We demonstrated attack with our own backdoored P, Q.

ScreenOS Version History

ScreenOS 6.1.0r7

- ANSI X9.31
- Seeded by interrupts
- Reseed every 10k calls
- 20-byte IKE nonces

ScreenOS 6.2.0r0 (2008)

- Dual EC $\rightarrow$ ANSI X9.31
- Reseed bug exposes raw Dual EC
- Reseed every call
- Nonces generated before keys
- 32-byte IKE nonces

- Attacker changed constant in 6.2.0r15 (2012).
- But passive decryption enabled in earlier release.
- Juniper’s "fix" was to reinstate original Q value. After our work they removed Dual EC completely.

September 2021: Publicly attributed to China

“Members of a hacking group linked to the Chinese government called APT 5 hijacked the NSA algorithm in 2012, according to two people involved with Juniper?s investigation and an internal document detailing its findings that Bloomberg reviewed. The hackers altered the algorithm so they could decipher encrypted data flowing through the virtual private network connections created by NetScreen devices. They returned in 2014 and added a separate backdoor that allowed them to directly access NetScreen products, according to the people and the document.”

`https://www.bloomberg.com/news/features/2021-09-02/`

`juniper-mystery-attacks-traced-to-pentagon-role-and-chinese-hackers`

Lessons and discussion

- Attacker repurposed cryptographically secure key escrow/backdoor with small change to source code that went unnoticed for years.
- Juniper's *original implementation* contained critical vulnerabilities that went unnoticed for years.

Fallout: Simon and Speck controversy

NSA introduced two “lightweight” ciphers in 2013.

Submitted them to ISO for standardization.

Criticized for having too small of a security margin.

Mistrust of NSA led to rejection by ISO working group, though they were later adopted by other ISO working groups.

2019: suspicions over Russian ciphers

Russia standardized Streebog hash function and Kuznyechik block cipher, with GOST.

Submitted them to ISO: “if any abroad citizen, company or governmental structure have a wish to cooperate with Russian information services they have to implement these algorithms. We hope that international standardization will make this implementation easier.”

Academics published articles finding several weaknesses.

Current Law Enforcement Access Debates

- 2016 Apple v. FBI
- Apple and Facebook CSAM detection algorithms.
- ...

Expectations for cryptographic design

How does an algorithm designer prove a lack of backdoors?

- Open analysis and standardization process.
- “Nothing up my sleeve” constants.
- Trust is a social process among humans.

Lessons and discussion

- Technical backdoors in our infrastructure don’t go away when the political environment changes.
- Cannot assign cryptography based on nationality.
- Added complexity of special access introduces unexpected vulnerabilities.