

# Applied Kleptography



Matthew Green  
Johns Hopkins University



[Download SuperFish Removal Tool](#)

## goto fail; // [Apple SSL bug](#) test site

This site will help you determine whether your computer is vulnerable to [#gotofail](#)

## Tracking the FREAK Attack



QUALYS® **SSL LABS**

The DROWN Attack



LOGJAM ATTACK (CVE-2015-4000)

TLS Vulnerability

## Response to improving security

- For the past decade, NSA has lead an aggressive, multi-pronged effort to break widely used Internet encryption technologies
- Cryptanalytic capabilities are now coming on line
- Vast amounts of encrypted Internet data which have up till now been discarded are now exploitable
- Major new processing systems, SIGDEV efforts and tasking must be put in place to capitalize on this opportunity

### PTD "We penetrate targets' defences."



This information is exempt from disclosure under the Freedom of Information Act 2000 and may be subject to exemption under other UK information legislation. Refer disclosure requests to GCHQ on 01242 221491 x30306 (non-sec) or email [infoleg@gchq](mailto:infoleg@gchq)

© Crown Copyright. All rights reserved.

# This talk

- Our community has made enormous progress towards building secure cryptographic systems...

# This talk

- **Our community has made enormous progress towards building secure cryptographic systems...**
  - At the level of algorithm and protocol design...
  - At the level of implementation and deployment...

# This talk

- Our community has made enormous progress towards building secure cryptographic systems...
  - At the level of algorithm and protocol design...
  - At the level of implementation and deployment...

**Yet all of this progress is based on the assumption that system designers are on our side.**

# This talk

- Our community has made enormous progress towards building secure cryptographic systems...
  - At the level of algorithm and protocol design...
  - At the level of implementation and deployment...

**Yet all of this progress is based on the assumption that system designers are on our side.**

**What if they aren't?**

# Kleptography

(n.) The study of stealing cryptographic secrets securely and subliminally.

(Young & Yung, 1996)

## Kleptography: Using Cryptography Against Cryptography

Adam Young\* and Moti Yung\*\*

**Abstract.** The notion of a Secretly Embedded Trapdoor with Universal Protection (SETUP) has been recently introduced. In this paper we extend the study of stealing information securely and subliminally from black-box cryptosystems. The SETUP mechanisms presented here, in contrast with previous ones, leak secret key information without using an explicit subliminal channel. This extends this area of threats, which we call "kleptography".

We introduce new definitions of SETUP attacks (strong, regular, and weak SETUPS) and the notion of  $m$  out of  $n$  leakage bandwidth. We show how to implement Kleptography in a black box environment. Finally, we

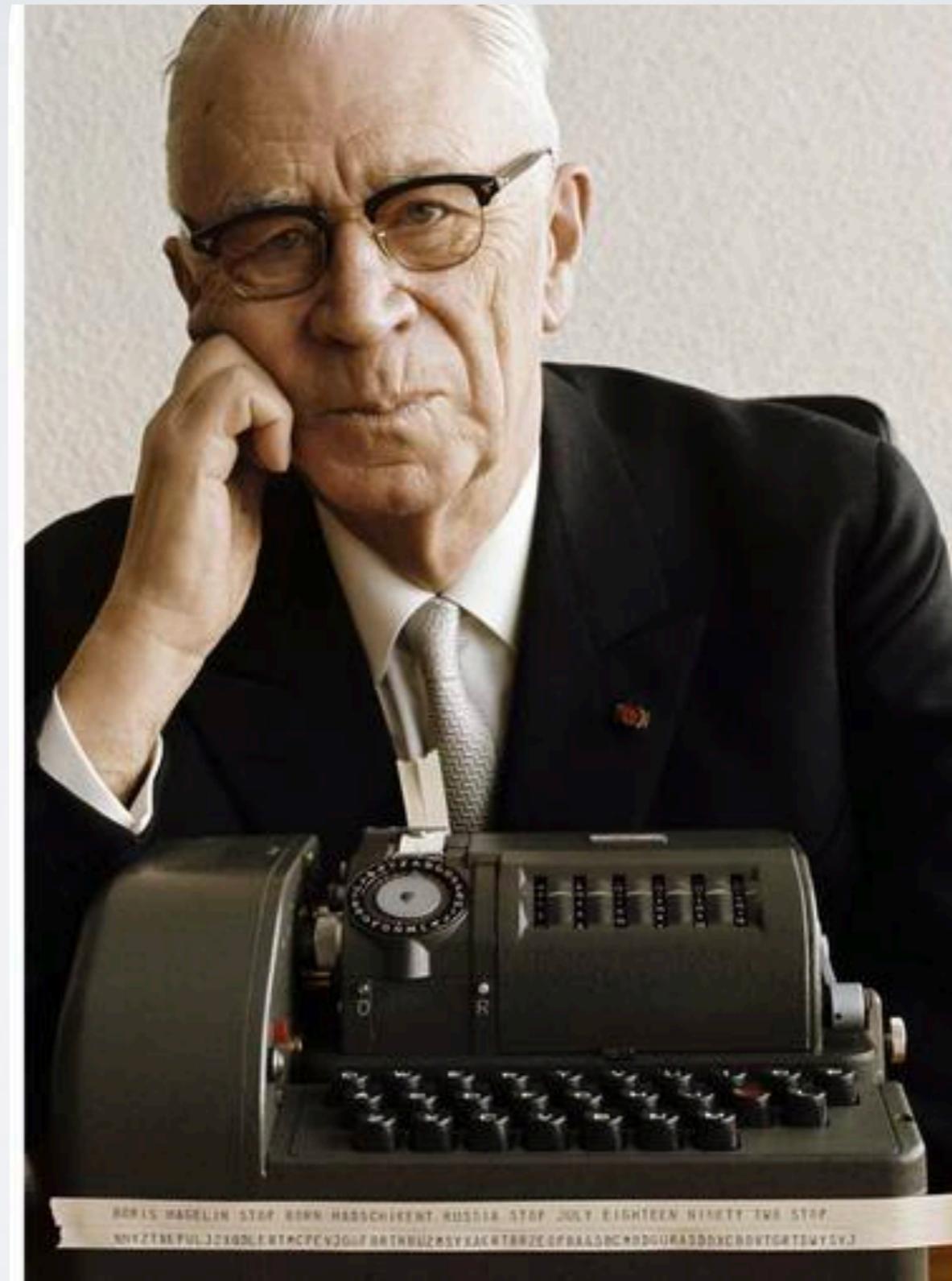
## The Dark Side of “Black-Box” Cryptography or: Should We Trust Capstone?\*

Adam Young<sup>\*\*</sup> and Moti Yung<sup>\*\*\*</sup> Yorktown Heights, NY 10598, USA.  
Email: moti@watson.ibm.com

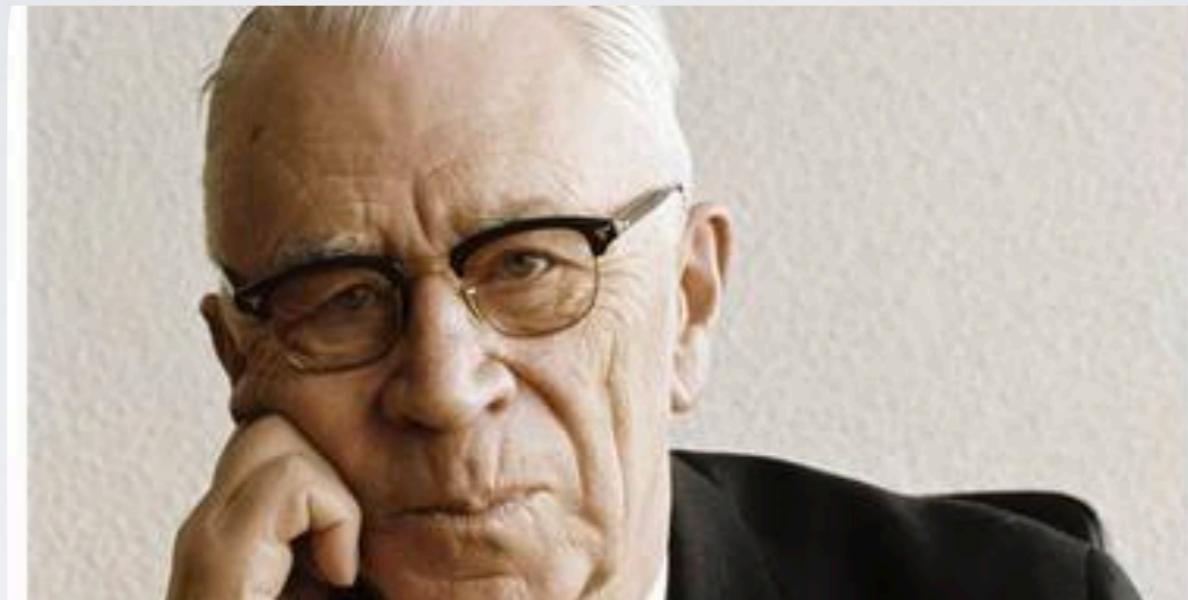
**Abstract.** The use of cryptographic devices as “black boxes”, namely trusting their internal designs, has been suggested and in fact Capstone technology is offered as a next generation hardware-protected escrow encryption technology. Software cryptographic servers and programs are being offered as well, for use as library functions, as cryptography gets more and more prevalent in computing environments. The question we address in this paper is how the usage of cryptography as a black box exposes users to various threats and attacks that are undetectable in a black-box environment. We present the SETUP (Secretly Embedded Trapdoor with Universal Protection) mechanism, which can be embedded in a cryptographic black-box device. It enables an attacker (the



# A bit of history (~1950s-1980s)



# A bit of history (~1950s-1980s)



~~REF ID: A60616~~  
~~TOP SECRET~~

EO 3.3(h)(2)  
PL 86-36/50 USC 3605

~~TOP SECRET~~

e. Hagelin Senior thanked me but said he did not need time to think the matter over; he was prepared to make an answer then and there to our proposal: He accepted it without any reservations or desire for modifications. But he would like to have some more details; he wanted to know, for example,

I told him I was not authorized nor able to give him more information on this score at this time but I had no doubt that such details would be forthcoming at the proper time.

## (U) COMPUTER NETWORK OPERATIONS (U) SIGINT ENABLING

Source: NYT/ProPublica

### (U) Project Description

(TS//SI//NF) The SIGINT Enabling Project actively engages the US and foreign IT industries to covertly influence and/or overtly leverage their commercial products' designs. These design changes make the systems in question exploitable through SIGINT collection (e.g., Endpoint, MidPoint, etc.) with foreknowledge of the modification. To the consumer and other adversaries, however, the systems' security remains intact. In this

- (TS//SI//NF) Shape the worldwide commercial cryptography marketplace to make it more tractable to advanced cryptanalytic capabilities being developed by NSA/CSS. [CCP\_00090]
  
- (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.

**(U) COMPUTER NETWORK OPERATIONS  
(U) SIGINT ENABLING**

Source: NYT/ProPublica

**(U) Project Description**

(TS//SI//NF) The SIGINT Enabling Project actively engages the US and foreign IT industries to covertly influence and/or overtly leverage their commercial products' designs. These design changes make the systems in question exploitable through SIGINT collection (e.g., Endpoint, MidPoint, etc.) with foreknowledge of the modification. To the consumer and other adversaries, however, the systems' security remains intact. In this

- (TS//SI//NF) Shape the worldwide commercial cryptography marketplace to make it more tractable to advanced cryptanalytic capabilities being developed by NSA/CSS. [CCP\_00090]

- (TS//SI//REL TO USA, FVEY) Insert vulnerabilities into commercial encryption systems, IT systems, networks, and endpoint communications devices used by targets.

**Insert vulnerabilities into commercial encryption systems, IT systems,**

- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.

**(U) COMPUTER NETWORK OPERATIONS  
(U) SIGINT ENABLING**

Source: NYT/ProPublica

**(U) Project Description**

(TS//SI//NF) The SIGINT Enabling Project actively engages the US and foreign IT industries to covertly influence and/or overtly leverage their commercial products' designs. These design changes make the systems in question exploitable through SIGINT collection (e.g., Endpoint, MidPoint, etc.) with foreknowledge of the modification. To the consumer and other adversaries, however, the systems' security remains intact. In this

- (TS//SI//NF) Shape the worldwide commercial cryptography marketplace to make it more tractable to advanced cryptanalytic capabilities being developed by NSA/CSS. [CCP\_00090]
- (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

**Influence policies, standards and specification for commercial public key technologies.**

- (TS//SI//REL TO USA, FVEY) Influence policies, standards and specification for commercial public key technologies.

**(U) COMPUTER NETWORK OPERATIONS  
(U) SIGINT ENABLING**

Source: NYT/ProPublica

**(U) Project Description**

(TS//SI//NF) The SIGINT Enabling Project actively engages the US and foreign IT industries to covertly influence and/or overtly leverage their commercial products' designs. These design changes make the systems in question exploitable through SIGINT collection (e.g., Endpoint, MidPoint, etc.) with foreknowledge of the modification. To the consumer and other adversaries, however, the systems' security remains intact. In this

- (TS//SI//NF) Shape the worldwide commercial cryptography marketplace to make it more tractable to advanced cryptanalytic capabilities being developed by NSA/CSS. [CCP\_00090]

To the consumer and other adversaries, however, the systems' security remains intact.

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.

# How do you build a Kleptographic system?

- **That is, a system many will use?**
  - Unlike Crypto AG, you can't mandate the hardware
  - The protocols are already extant (IPSec, SSL, etc.)
  - Can't really mandate the software
  - You can mandate cryptographic algorithms
  - You can validate cryptographic implementations

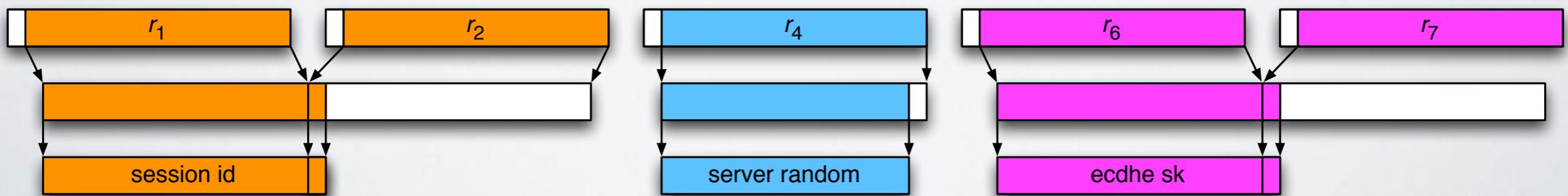
# Achilles heel: randomness

- **Many protocols, one commonality:**
  - Most cryptographic protocols devour random bits
    - Ex: 108 bytes / TLS session (ECDH+ECDSA, server)
  - The quality of those bits is hugely important
  - Attacker who can predict (P)RNG output can break (almost) any protocol



# Achilles heel: randomness

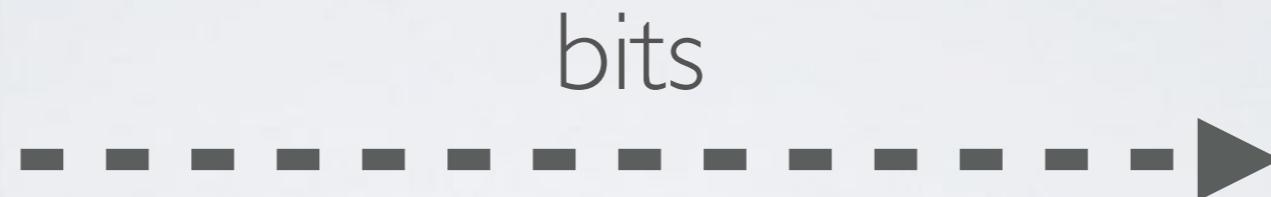
- **Moreover, a single generator may produce both public and secret values**
  - In practice an RNG must remain secure when the attacker can see some public output
  - This is something engineers take for granted, and rely on w/o conscious thought



# RNG System Architecture (I)



**TRNG**



**Crypto  
Protocols**

(SSL, TLS,  
IPSec, etc.)

Probabilistic:  
system specific,  
hardware/entropy  
collection

# RNG System Architecture (2)



**TRNG**

seed  
→



**PRNG  
(DRBG)**

bits  
→

**Crypto  
Protocols**

(SSL, TLS,  
IPSec, etc.)

Probabilistic:  
system specific,  
hardware, entropy  
collection

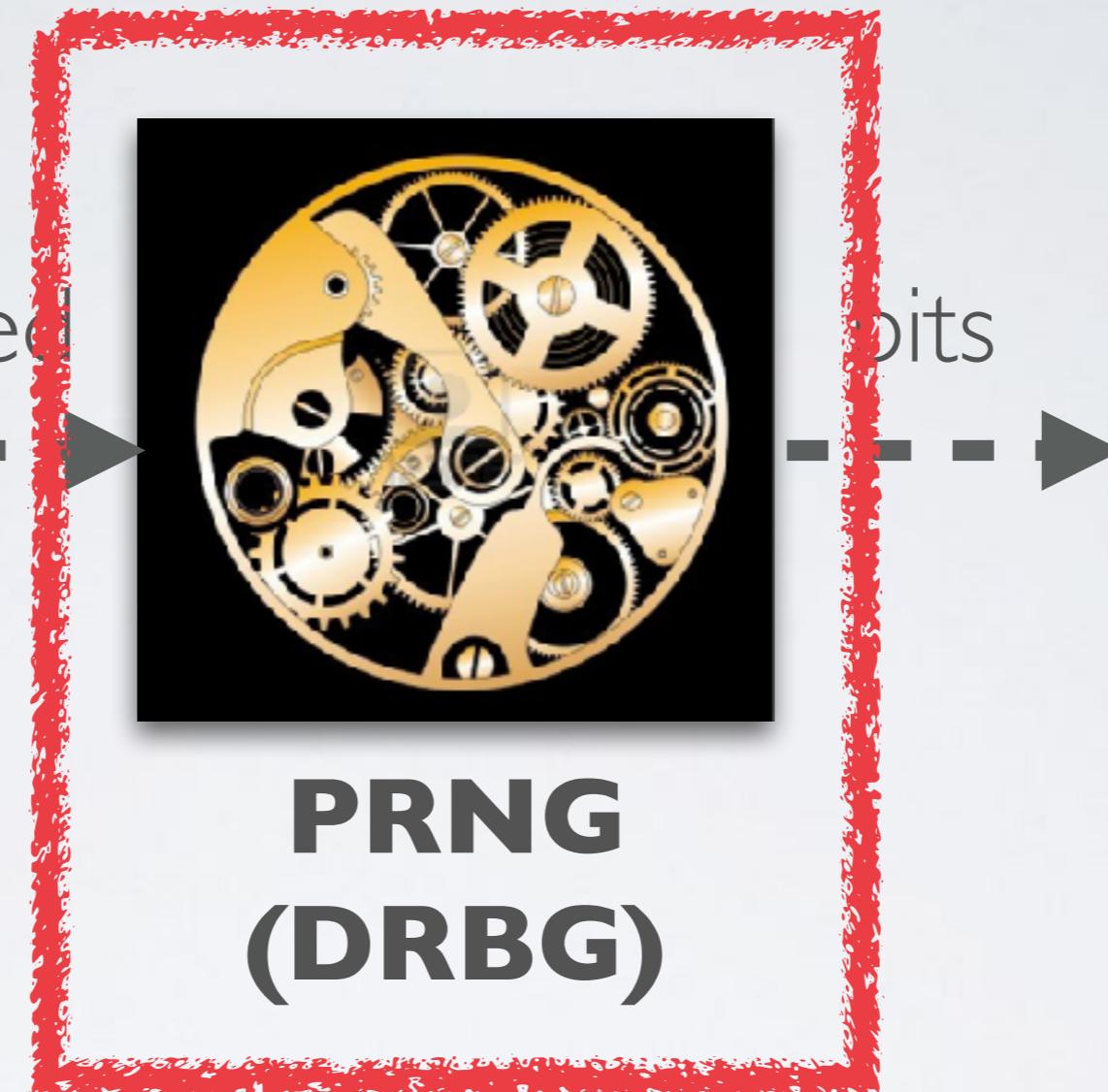
Deterministic:  
computational, fast,  
algorithmic

# RNG System Architecture (2)



**TRNG**

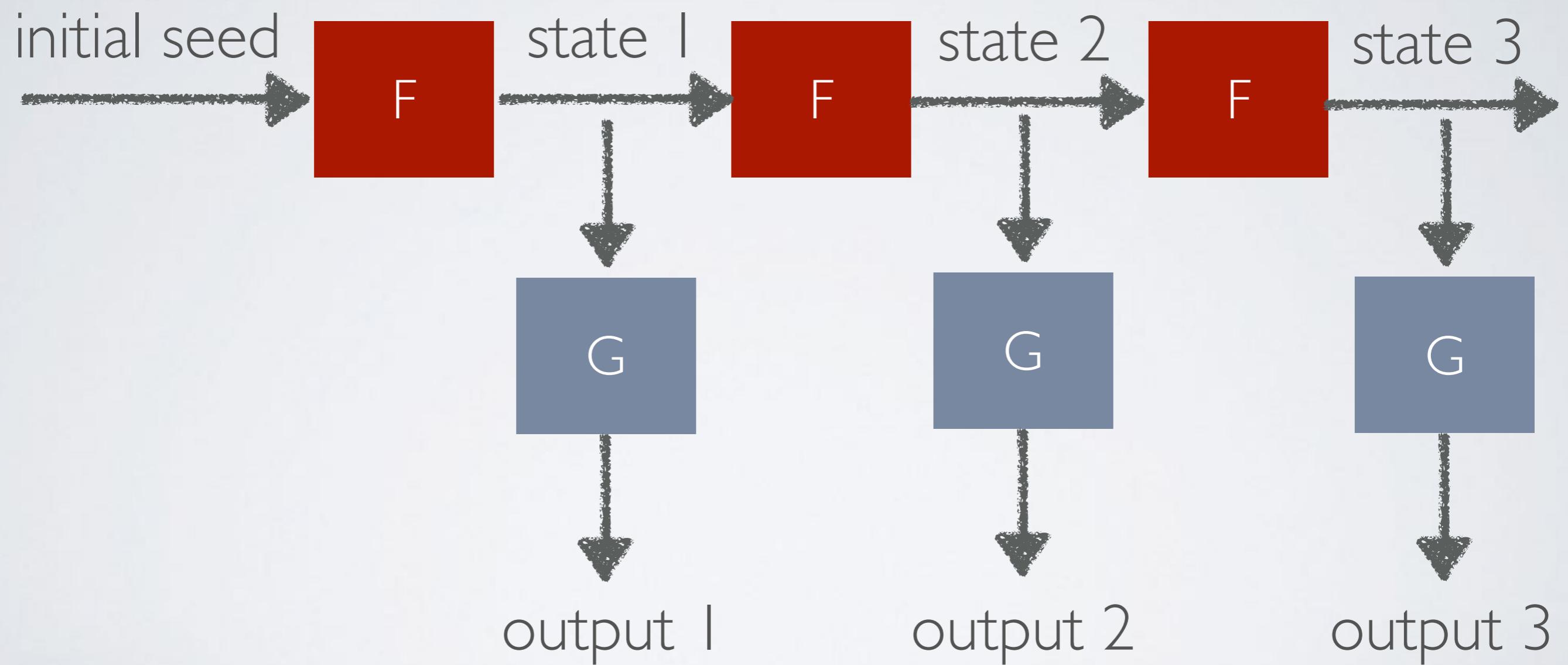
Probabilistic:  
system specific,  
hardware, entropy  
collection



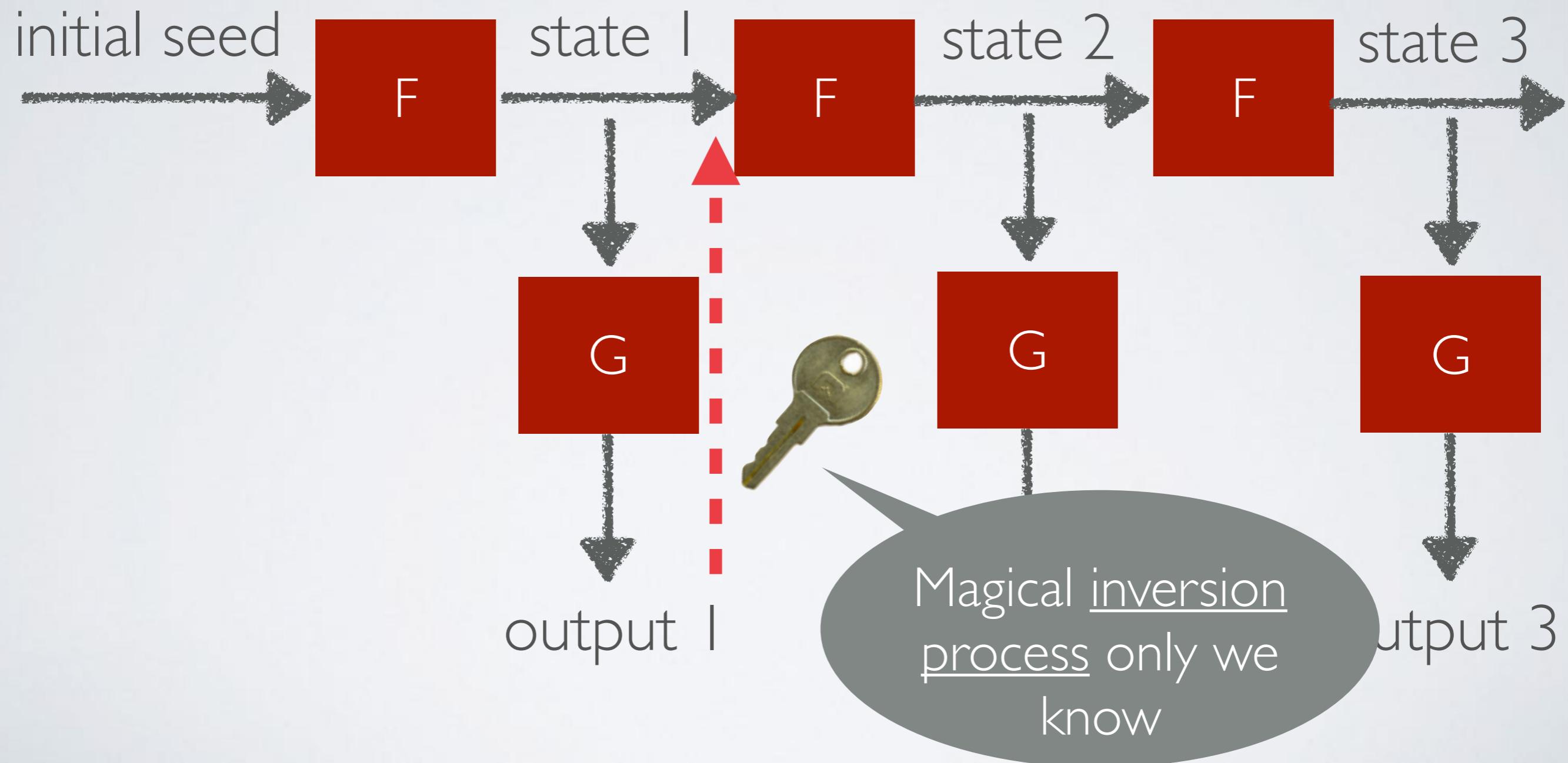
Deterministic:  
computational, fast,  
algorithmic

**Crypto  
Protocols**  
(SSL, TLS,  
IPSec, etc.)

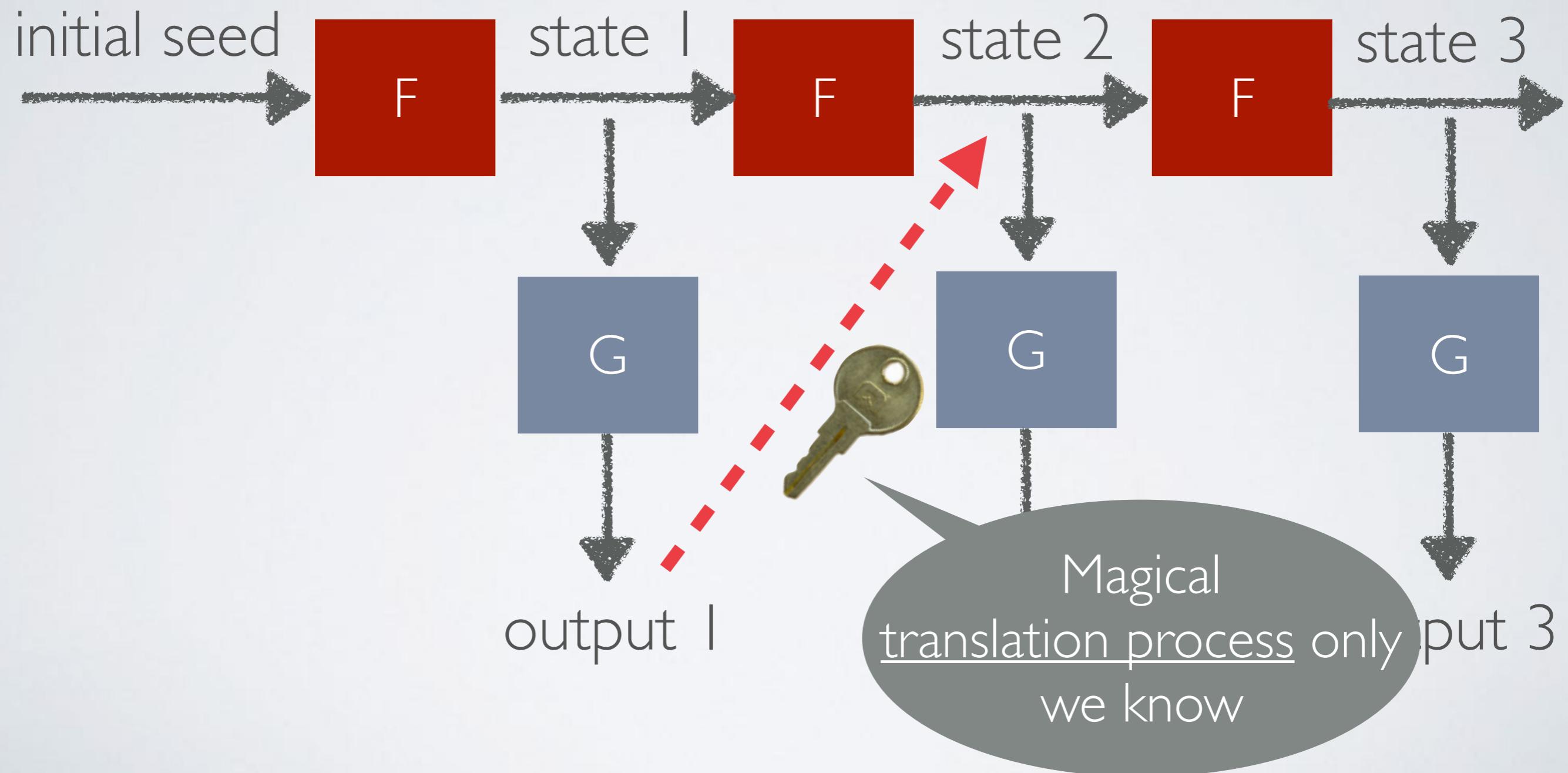
# Template for a DRBG



# Kleptographic proposal #1: Make G invertible



# Kleptographic proposal #2: A mapping from G to F



# 1996: Young & Yung SETUPs

**Schnorr.KG:**  $\langle g \rangle = \mathbb{G}$  of prime order  $q$

$$sk = x \leftarrow \mathbb{Z}_q, PK = g^x$$

**Schnorr.Sign(I):**  $k \leftarrow \mathbb{Z}_q$  output: (r, s)  
 $c = H(PK \| g^k \| m)$   
 $r = g^k$   
 $s = xc + k \bmod q$

# Young & Yung: SETUPs

**SETUP.KG:**  $\text{MK} = g^y, msk = y$

**Schnorr.KG:**  $\langle g \rangle = \mathbb{G}$  of prime order  $q$

$$sk = x \leftarrow \mathbb{Z}_q, PK = g^x$$

**Schnorr.Sign(I):**  $k \leftarrow \mathbb{Z}_q$  output: (r, s)

$$c = H(PK \| g^k \| m)$$
$$r = g^k$$
$$s = xc + k \bmod q$$

# Young & Yung: SETUPs

**SETUP.KG:**  $\text{MK} = g^y, msk = y$

**Schnorr.KG:**  $\langle g \rangle = \mathbb{G}$  of prime order  $q$

$$sk = x \leftarrow \mathbb{Z}_q, PK = g^x$$

**Schnorr.Sign(1):**  $k \leftarrow \mathbb{Z}_q$  output: (r, s)

$$c = H(PK \| g^k \| m)$$

$$r = g^k$$

$$s = xc + k \bmod q$$

**Schnorr.Sign(2):**  $k' \leftarrow H(\text{MK}^k) \in \mathbb{Z}_q$

Compute next  
signature using  $k'$

# Young & Yung: SETUPs

**SETUP.KG:**  $\text{MK} = g^y, msk = y$

**Schnorr.KG:**  $\langle g \rangle = \mathbb{G}$  of prime order  $q$

Given  $r$  and  $msk$  we can recover  $k'$  as:

$$k' = H(r^{msk})$$

**Schnorr** and thus obtain the long term secret key. t: (r, s)

$$c = H(PK \| g^r \| m)$$

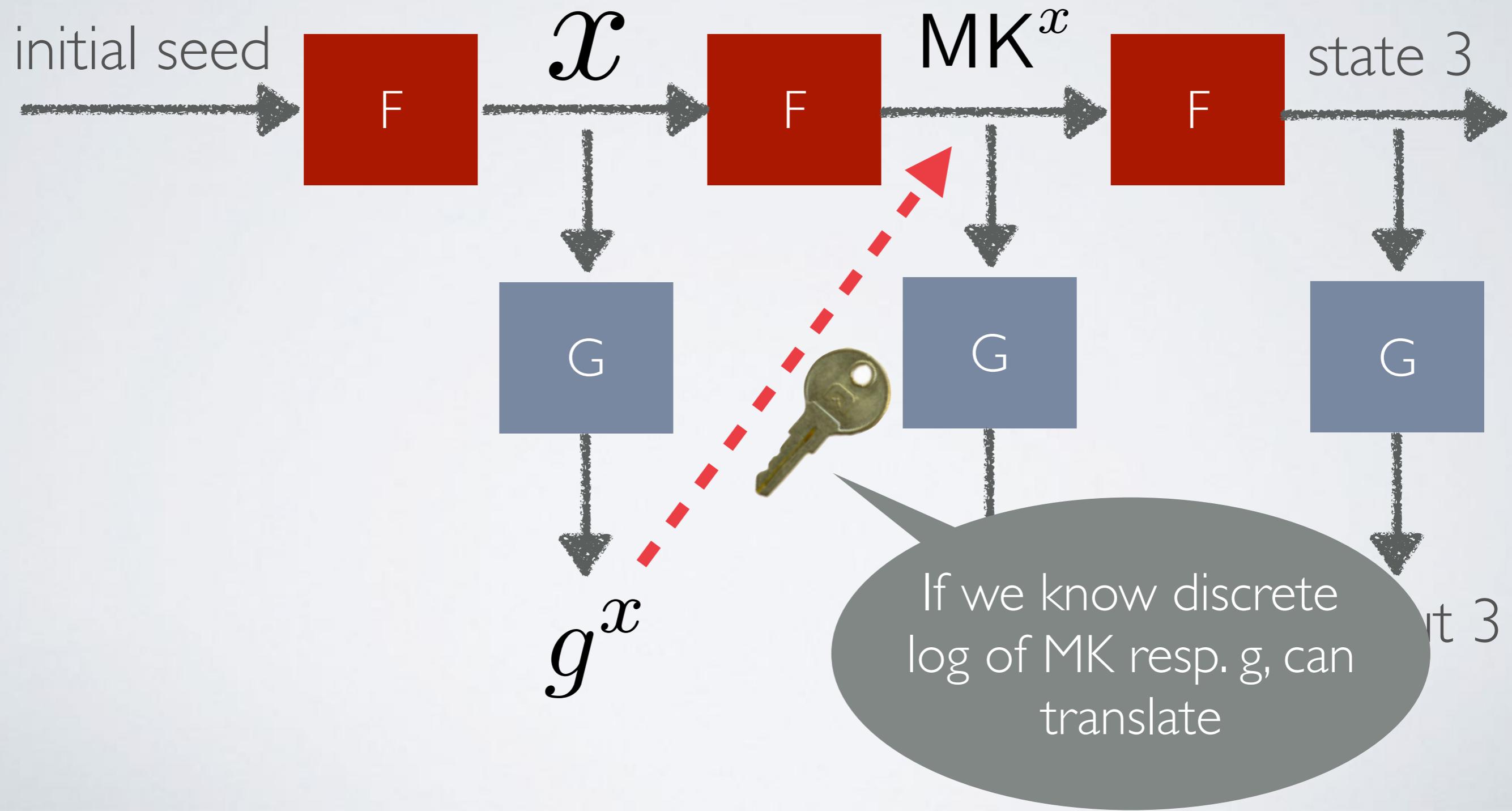
$$r = g^k$$

$$s = xc + k \bmod q$$

**Schnorr.Sign(2):**  $k' \leftarrow H(\text{MK}^k) \in \mathbb{Z}_q$

Compute next  
signature using  $k'$

# SETUP



# Dual EC DRBG

initial seed   $x$    $xP$   state 3

If you move this design into the EC setting & add some truncation, you get **Dual EC DRBG**.

Standardized by NIST in SP800-90A

(using prime-order elliptic curve subgroups).  
Vulnerability publicized by Shumow and Ferguson '07.

$xQ$

log of MK resp. g, can  
translate



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

By NICOLE PERLROTH, JEFF LARSON and SCOTT SHANE

Published: September 5, 2013 |  1466 Comments

Cryptographers have long suspected that the agency planted vulnerabilities in a standard adopted in 2006 by the National Institute of Standards and Technology and later by the International Organization for Standardization, which has 163 countries as members.

Classified N.S.A. memos appear to confirm that the fatal weakness, discovered by two Microsoft cryptographers in 2007, was engineered by the agency. The N.S.A. wrote the standard and aggressively pushed it on the international group, privately calling the effort “a challenge in finesse.”

“Eventually, N.S.A. became the sole editor,” the memo says.

# Fast forward to 2015

Next several slides, joint work with:

Stephen Checkoway, Jacob Maskiewicz, Christina Garman, Joshua Fried, Shaanan Cohney, Nadia Heninger, Ralf-Philipp Weinmann, Eric Rescorla, Hovav Shacham



Juniper is committed to maintaining the integrity and security of our products and wanted to make customers aware of critical patched releases we are issuing today to address vulnerabilities in devices running ScreenOS® software.

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. Once we identified these vulnerabilities, we launched an investigation into the matter, and worked to develop and issue patched releases for the latest versions of ScreenOS.

At this time, we have not received any reports of these vulnerabilities being exploited; however, we strongly recommend that customers update their systems and apply the patched releases with the highest priority.

# CVE-2015-7756

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](#).

**VPN Decryption (CVE-2015-7756) may allow a knowledgeable attacker who can monitor VPN traffic to decrypt that traffic.**

# Vulnerable -> Patched

## **ScreenOS 6.3.0r20** (vulnerable)

```
2551....9585320EEAF81044F20D5503  
0A035B11BECE81C785E6C933E4A8A131  
F6578107....interrupt disabled a  
2551....2c55e5e45edf713dc43475ef  
fe8813a60326a64d9ba3d2e39cb639b0  
f3b0ad10....interrupt disabled a
```

## **ScreenOS 6.3.0r21** (patched)

Sources: [Adam Caudill](#), Peter Bowen, HD Moore, Ralf Phillip Weinmann

# Dual EC DRBG

P-256 Weierstraß  $b$   
5AC635D8AA3A93E7B3EBBD55/69886BC651D05B  
P-256  $P$  x coord C3E27D2604B  
6B17D1F2E12C4247F8BCE6E563A440F277037D812DE633A0F4A139  
P-256 field order  
FFFFFFF00000000FFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551  
bad: 9585320EEAF81044F20D55030A035B11BECE81C785E6C933E4A8A131F6578107  
good: 2c55e5e45edf713dc43475effe8813a60326a64d9ba3d2e39cb639b0f3b0ad10  
nist:c97445f45cdef9f0d3e05e1e585fc297235b82b5be8ff3efca67c59852018192

NIST SP 800-90A

January 2012

**NIST Special Publication 800-90A**

**Recommendation for Random Number  
Generation Using Deterministic  
Random Bit Generators**

## FIPS Approved Algorithms

The following FIPS approved algorithms are supported:

- DSA , ECDSA Sign Verify
- SHA-1, SHA-256
- Triple-DES (CBC)

*Juniper Networks SSG 5 and SSG 20 Security Policy*

## *FIPS 140-2 SECURITY POLICY*

*Juniper Networks, Inc.*

*SSG 5 and SSG 20*

*HWP/N SSG-5 and SSG-20, FW Version ScreenOS 6.2.0*

*Document # 530-023728-01*

**Juniper doesn't  
appear to use Dual  
EC...**

- AES (CBC)
- HMAC-SHA-1, HMAC-SHA-256
- RSA Sign/Verify (PKCS #1)
- ANSI X9.31 DRNG

# Dual EC in ScreenOS

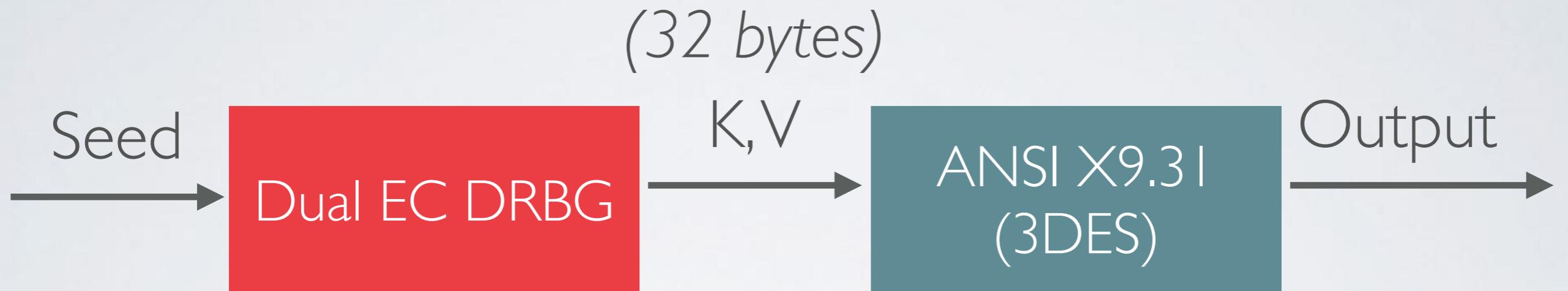
The following product families do utilize Dual\_EC\_DRBG, but do not use the pre-defined points cited by NIST:

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 does make use of the Dual\_EC\_DRBG standard, but is designed not to use Dual\_EC\_DRBG as its primary random number generator. ScreenOS uses it in a way that shouldn't be vulnerable to the possible issue that has been brought to light.” (2013)**

# RNG Cascade



**This approach  
should neutralize  
any backdoor**

```
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(); // Call to 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 }
```

Calls Dual EC to fill a buffer

Credit: William Pinckaers

```
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 }
```

“Runs” ANSI generator in place



```
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", 110);
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 }
```

Generates Dual EC output  
Sets prng\_output\_index = 32

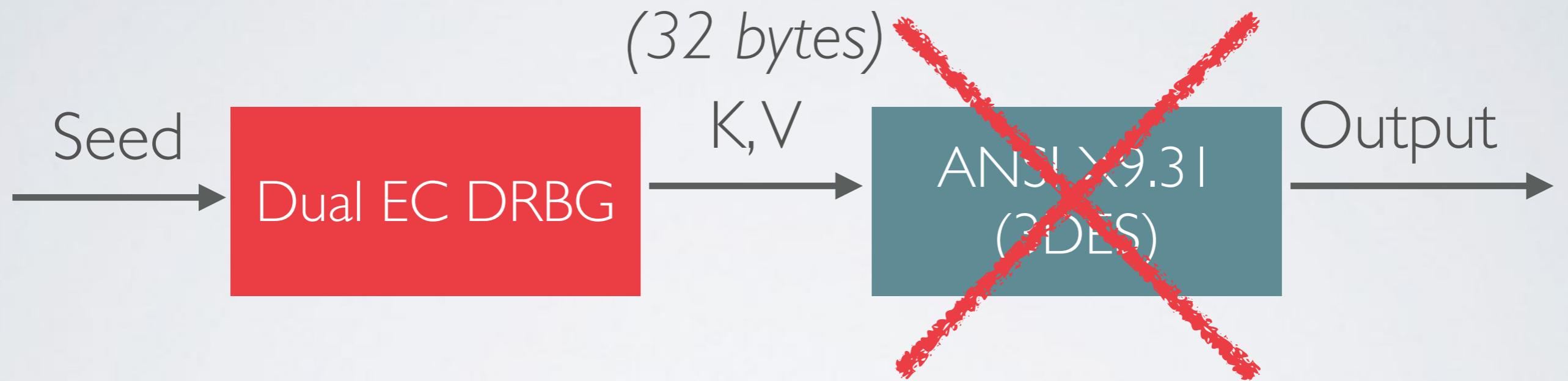
Credit: William Pinckaers

```
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 }
```

ANSI generator is never run.  
Dual EC output emitted.

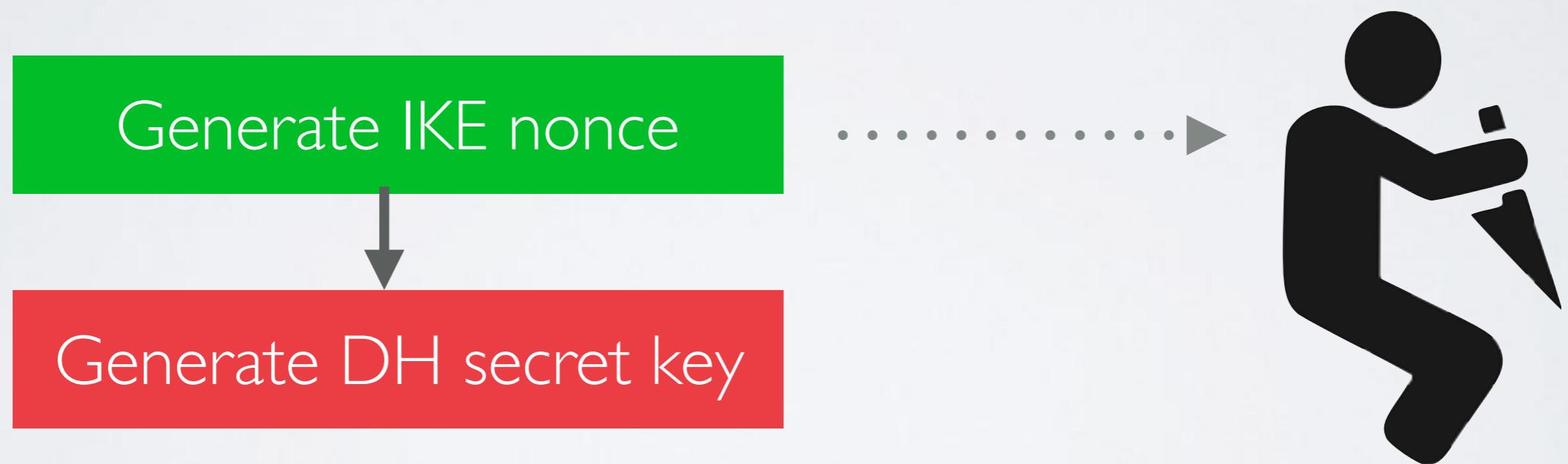


# Revised Cascade



# Exploiting IKE (Ideal)

- Like many protocols, outputs *nonces*
  - In ScreenOS 6.1 (pre-Dual EC): 20 bytes  
In ScreenOS 6.2 (with Dual EC): 32 bytes  
( $\geq 28$  bytes is sufficient to recover Dual EC state)



recompute DH secret key

# Exploiting IKE (Ideal)

- L
- This is (apparently) not what Juniper does



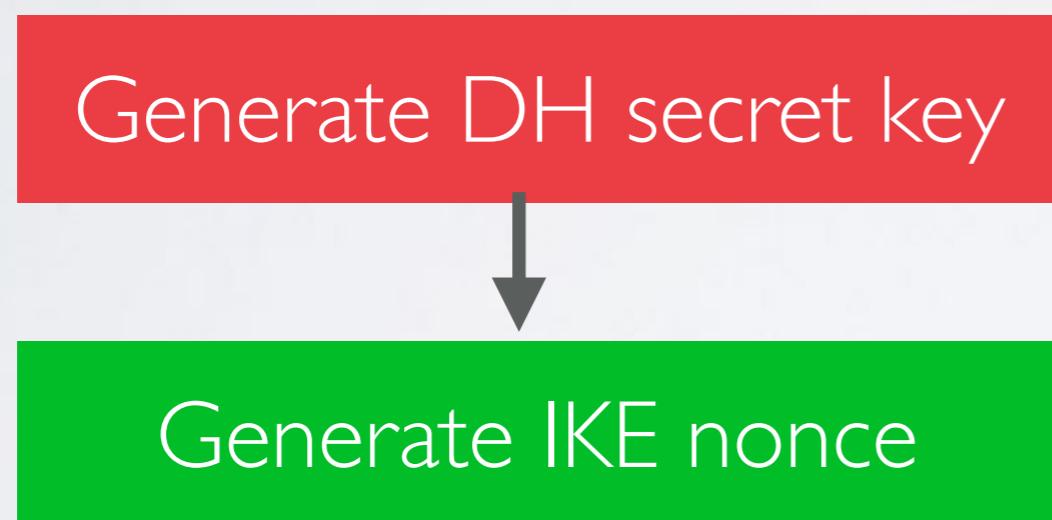
Generate DH secret key



recompute DH secret key

# Exploiting IKE (ScreenOS 6.1)

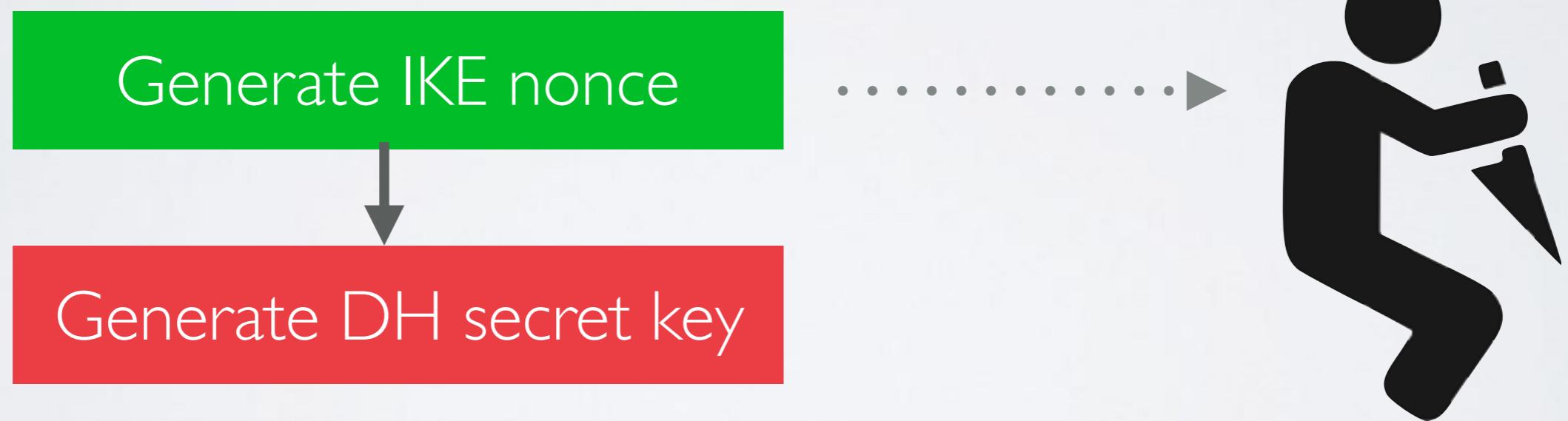
- All versions of ScreenOS appear to generate key first  
Nonce second
  - Even with Dual EC, hinders the attack



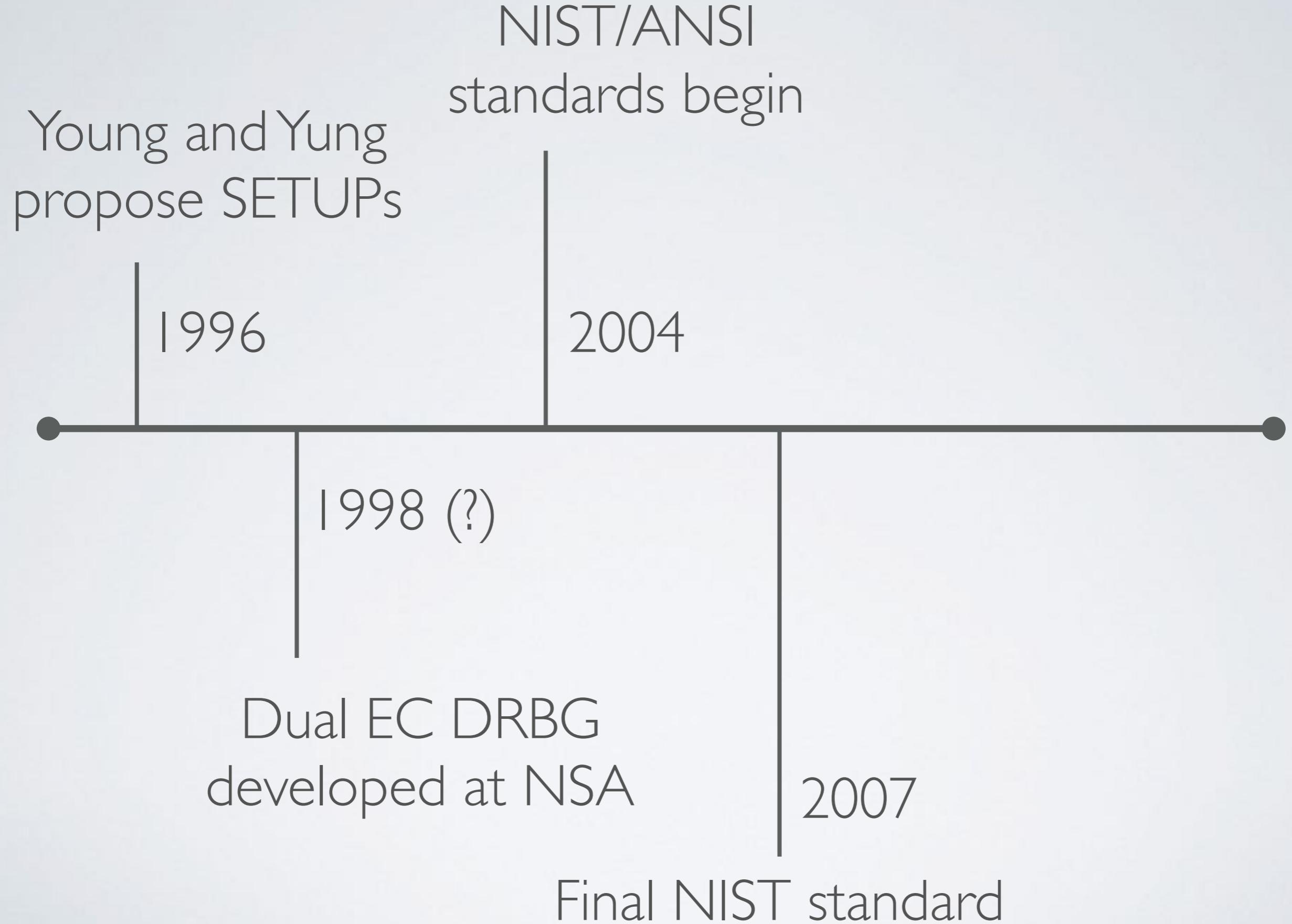
(must wait for next handshake)

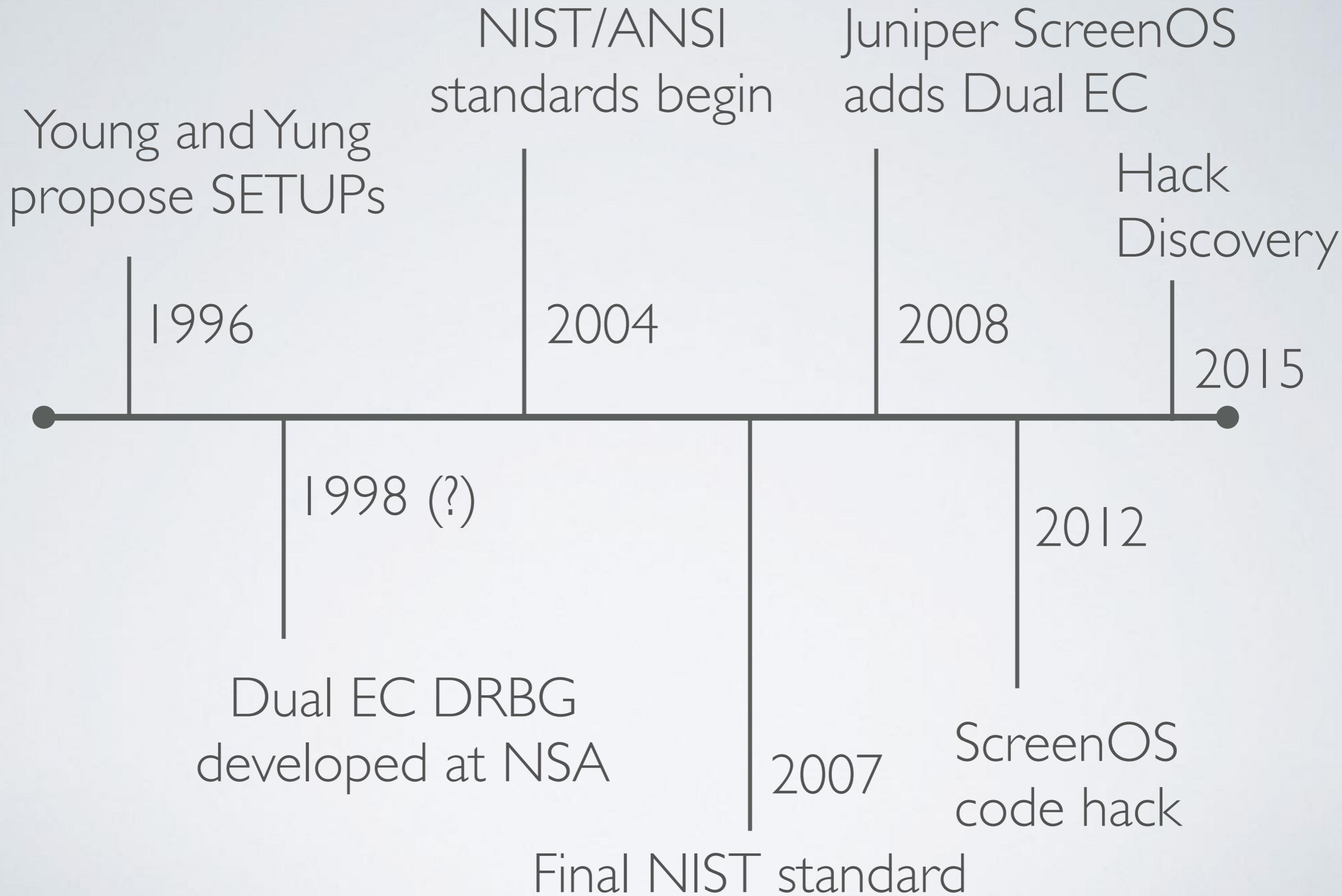
# Exploiting IKE (ScreenOS 6.2)

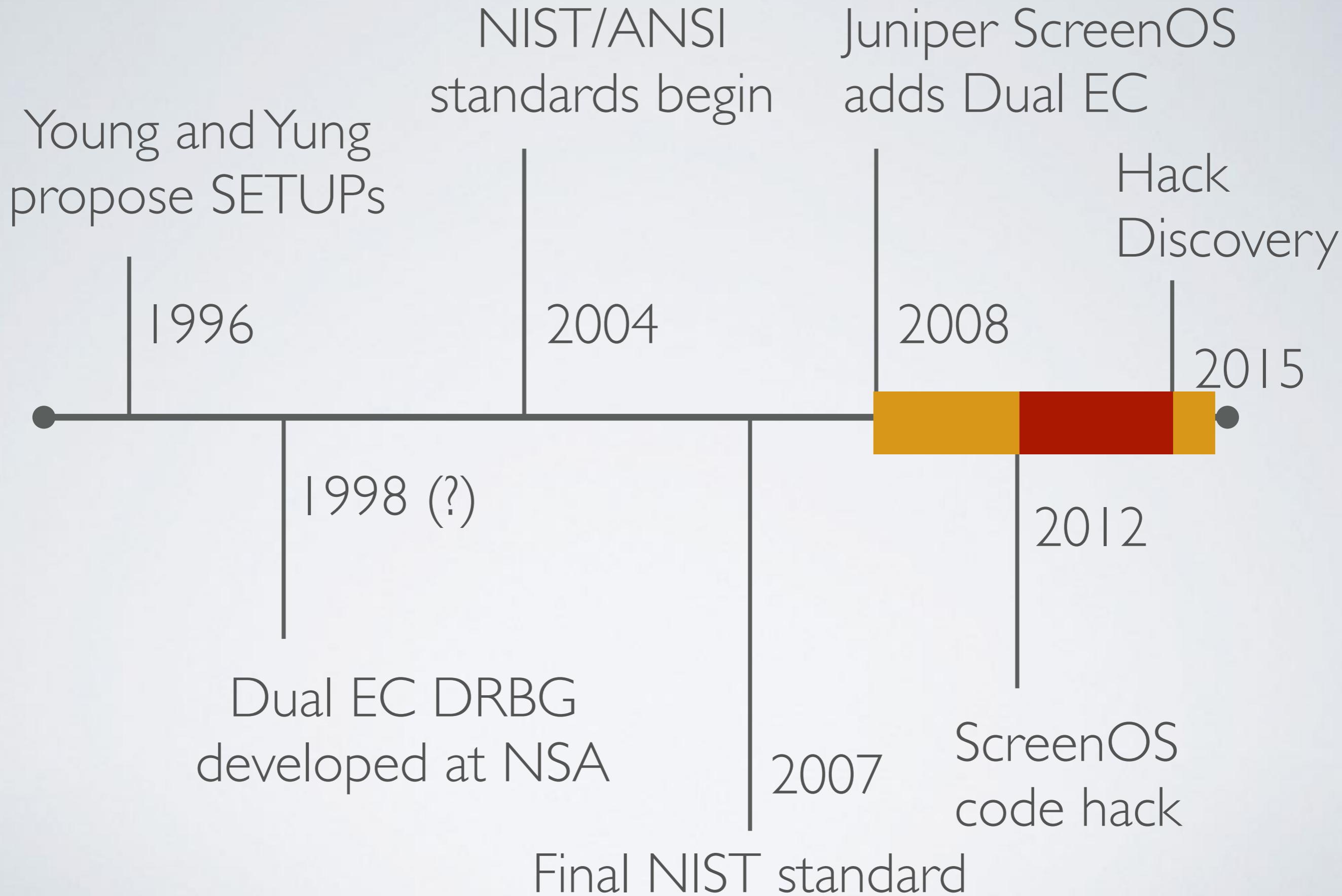
- ScreenOS 6.2 (the version that adds Dual EC)
  - Adds a *nonce pre-generation queue*
  - Effectively ensures that nonces are always generated first



recompute DH secret key









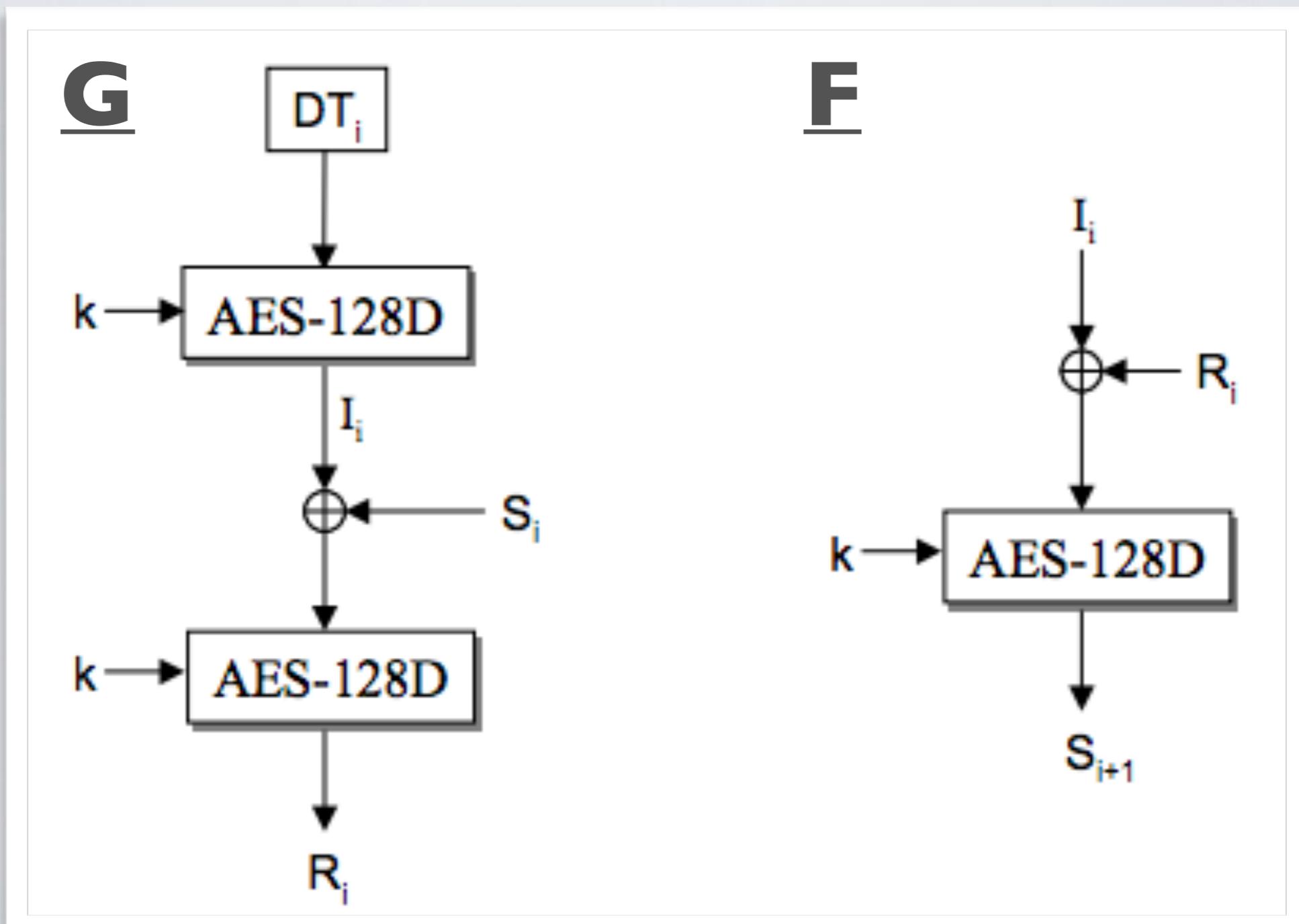
## **Question:**

Let's assume the Dual EC DRBG flaws were deliberate, not an accident.

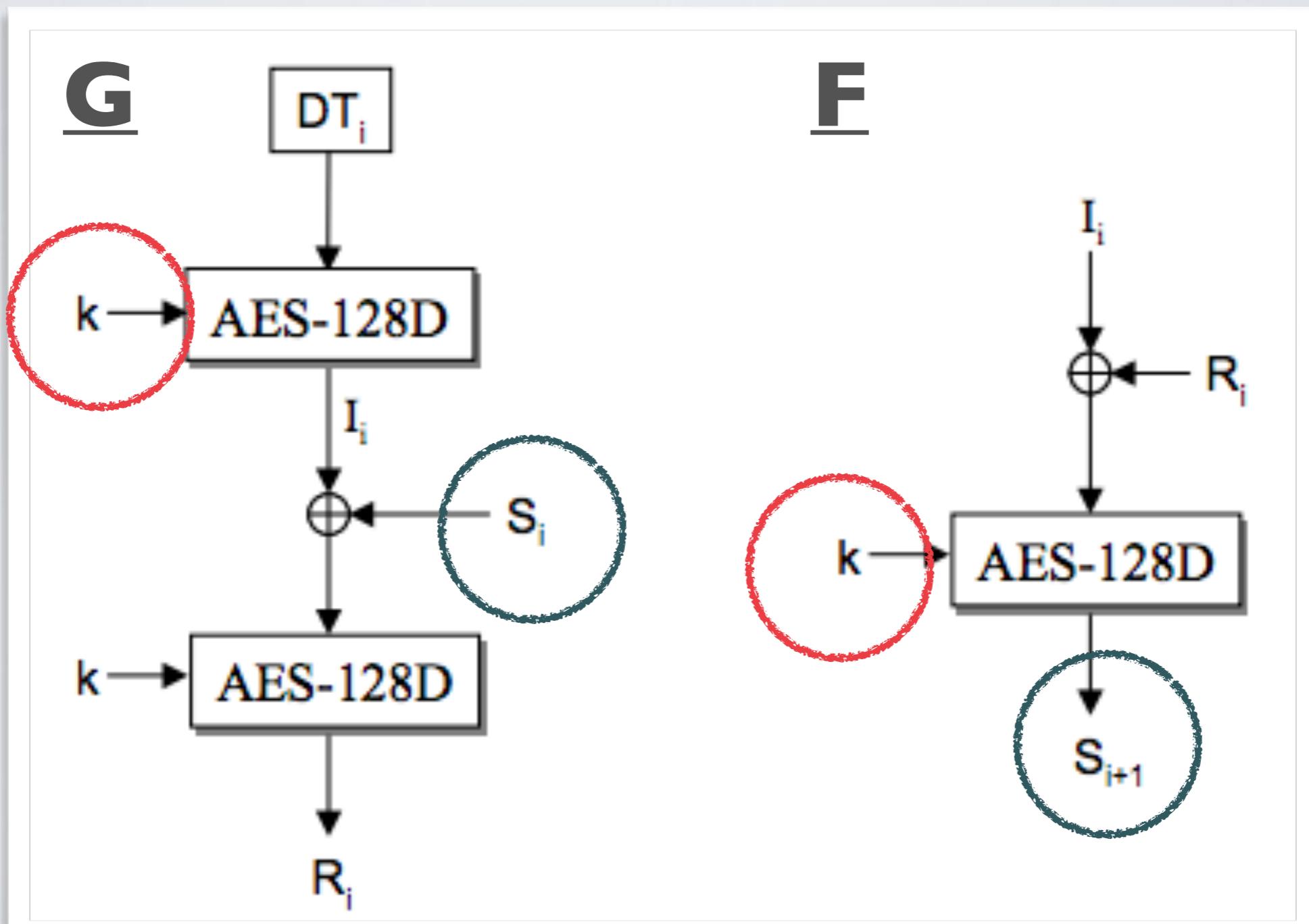
Let's assume that there is exists policy to promote vulnerabilities in VPN devices.

How would you implement kleptographic systems before SETUPs?

# ANSI X9.31



# ANSI X9.31



# Attacking ANSI X9.31

- **Most common DRBG (PRG) in FIPS devices**
  - Well-known vulnerability: Kelsey, Schneier, Wagner and Hall
    - Attacker who knows the key **k** (but not the seed **S**) can recover internal generator state from 16-32 out bytes
    - Key is never updated
    - Standard says nothing about this
    - To an attacker without knowledge of **k**, output is indistinguishable from random

**Security Target for the Fortinet FortiGate™-200B  
and 620B Unified Threat Management Solution  
and FortiOS 4.0 CC Compliant Firmware: EAL4+**

The storage area for private cryptographic keys, plaintext cryptographic keys and all other critical cryptographic security parameters is a flash RAM device. Zeroization of these storage areas occurs when the Security Administrator executes a factory reset or enables FIPS-CC mode. At these times, all non-hard-coded keys and critical security parameters are zeroized by lifting the voltage from the bits comprising the key, which has the same effect as overwriting the storage area with zeroes. The hard-coded keys are ANSI X9.31 RNG AES Key, Firmware update key, configuration integrity key, configuration backup key.

*Prepared by:*

**Electronic Warfare Associates-Canada, Ltd.**  
55 Metcalfe St., Suite 1600  
Ottawa, Ontario  
K1P 6L5

# Exploiting FortiOS

- **Reverse-engineered FortiOS implementation**
  - With Shaanan Cohney and Nadia Heninger
  - Not a trivial attack to implement: requires guessing a microsecond-level timestamp value updated at each block
  - By adjusting granularity of this timer, can make the attack cost  $2^{40}$  or  $2^{50}$  AES operations (and up)
  - Many optimizations. Full recovery of Diffie-Hellman private keys from a protocol transcript in about 15 seconds

# Summing up

- **Catastrophic RNG vulnerabilities in 2 major VPN device manufacturers during the same time period**
  - Hanlon's razor: "Never attribute to **malice** that which is adequately explained by **stupidity**"
  - Heinlein's Razor: "Never attribute to **malice** that which can be adequately explained by **stupidity**, but don't rule out **malice**."

So how do we fix this?

# We can't do everything

- **Basic problem: if adversary has unlimited control over device implementation, we lose**
  - E.g., perfectly correct implementation + exfiltrate 256 bit PRG state through some timing channel
  - But in the main, the adversary is constrained by two factors:
    1. Complexity of the modifications (code does get reviewed!)
    2. Effect on the protocol transcript (can be monitored)
  - This explains why corrupted RNG designs are so popular

# I. Build more resilient protocols

- **Example: PSK in IKEv1**

- PSK is fed into the KDF
- If PSK is high entropy, devices not exploitable!

# I. Build more resilient protocols

- **Example: PSK in IKEv1**

- PSK is fed into the KDF
- If PSK is high entropy, devices not exploitable

- **Example: PSK in IKEv2**

- PSK is not fed into the KDF
- Devices may be exploitable!

# 2. Replace FIPS validation

- **FIPS validation does not work**

- Each of the devices I've discussed went through high-level FIPS (CMVP) validation, some at high EAL levels!
- All the preceding vulnerabilities should have been caught
- FIPS validation == alg tests + compliance
- Worse, the FortiOS hard-coded key was a testing key
  - Why is there a testing key in the device?
  - Because FIPS mandates runtime tests!

# 2a. Whole-protocol evaluation

- **Validate devices by speaking their language**
  - Rather than testing individual algorithms, run live tests with the device
  - Protocol should complete correctly with a testing endpoint
  - Now:
    - 1. Have device prove the correctness of its protocols using efficient 2PC (many challenges!)
    - 2. Fuzz firmware to identify hard-coded parameters

## 2b. Full formal verification

- **Formally verify the entire DRBG stack**
  - Joint work with Andrew Appel, Katherine Ye, Lennart Beringer: developed a formal proof of security in Coq/FCF
  - **Step 1:** Machine-prove that the NIST HMAC-DRBGs are actually PRGs
  - **Step 2:** Machine-prove that mbed-TLS (C) stack implements the specification (including HMAC and SHA — already done)
  - **Step 3:** Link the proofs together

# This should drive our research

- **Mostly it doesn't. But some notable exceptions:**
  - Algorithm Substitution Attacks (Bellare, Paterson, Rogaway)
  - Kleptography (Young, Yung)
  - Formal Treatments of RNGs (Dodis et al.)
  - Formal Verification Approaches (INRIA, MSR, Princeton)



I WANT TO  
BELIEVE