# The Entropy Bogeyman

Ed Morris and Khai Van
November 5, 2015
International Crypto Module Conference

Gossamer
Laboratories

# Topics

- Overview
- Background
- Design Problems
- Public Entropy Vulnerabilities
- Recommendations

# Overview

- **Entropy:**
  - **measure of the unpredictability of a string of bits**
- **Entropy underpins cryptography**
- **Poor entropy can undermine secu**

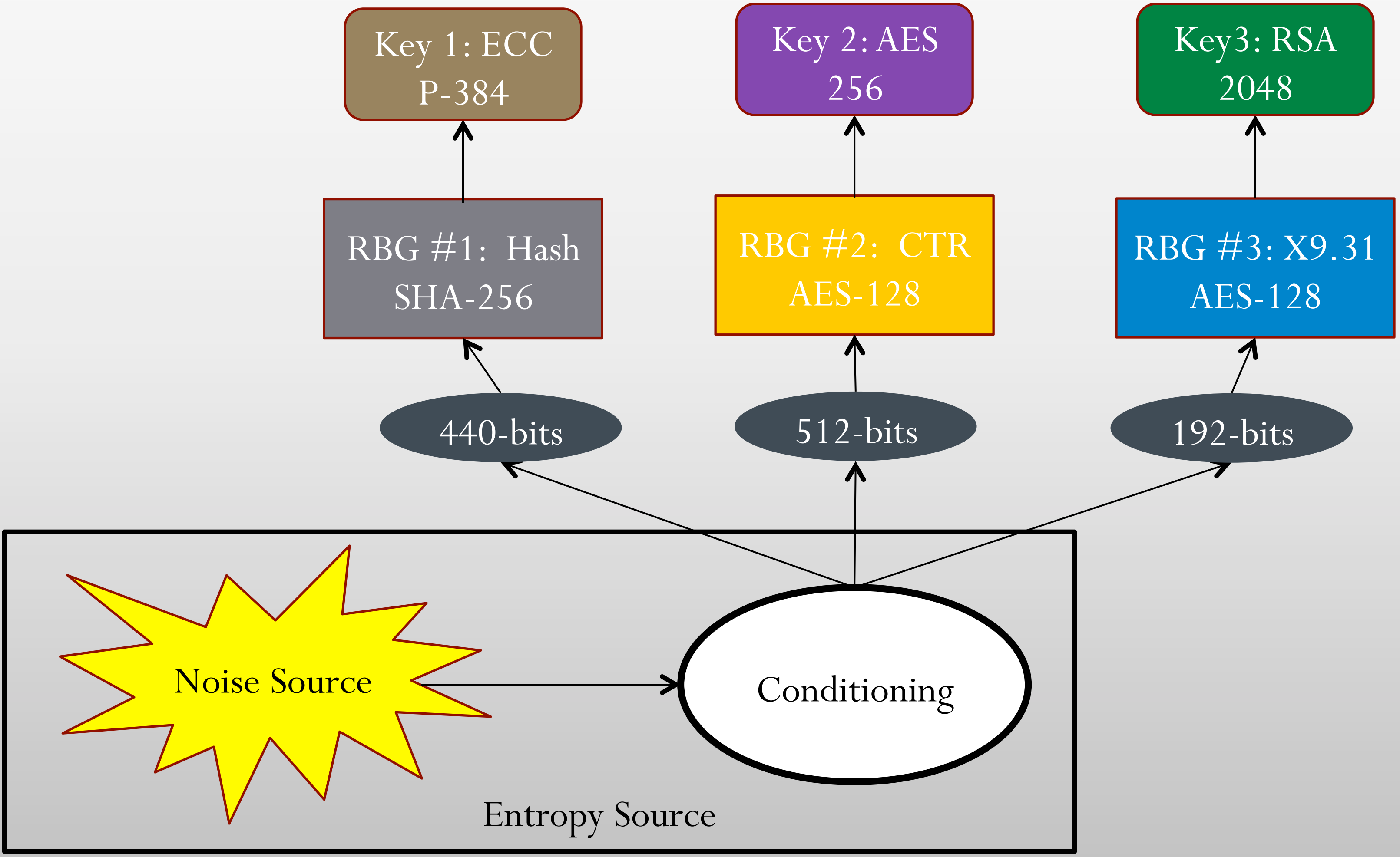
➢ **But to what degree?**

➢ **And with what consequences?**

# Background (Entropy Defined)

- Measures of entropy:
  - Shannon Entropy, Renyi, Min-entropy, etc.
- Statistical assessments for entropy:
  - DIEHARD
  - FIPS 140-1 (11 Jan 1994)
  - AIS 20 (2 Dec 1999)
  - NIST SP 800-22 (1 Dec 2000)
  - AIS 20/31 (18 Sep 2011)
  - NIST SP 800-90B (Aug 2012)
- Assessing entropy is non-trivial

# Background (Entropy Arch.)



Key 1: ECC P-384

Key 2: AES 256

Key3: RSA 2048

RBG #1: Hash SHA-256

RBG #2: CTR AES-128

RBG #3: X9.31 AES-128

440-bits

512-bits

192-bits

Noise Source

Conditioning

Entropy Source

# Background (noise & whitening)

- **Source type (hardware or software)**
  - Hardware: ring oscillators, voltage oscillators
  - Software: timing variations in high-precision clock
- **Types of whitening/conditioning**
  - Unbiasing (Von Neumann)
  - Condensing (XORing, folding)
  - Pool "stirring" (Hashing)
  - Linear Feedback Shift Register (LFSR)
- **Assess min-entropy of raw entropy samples**
  - Tim Hall's SP800-90B Python Suite

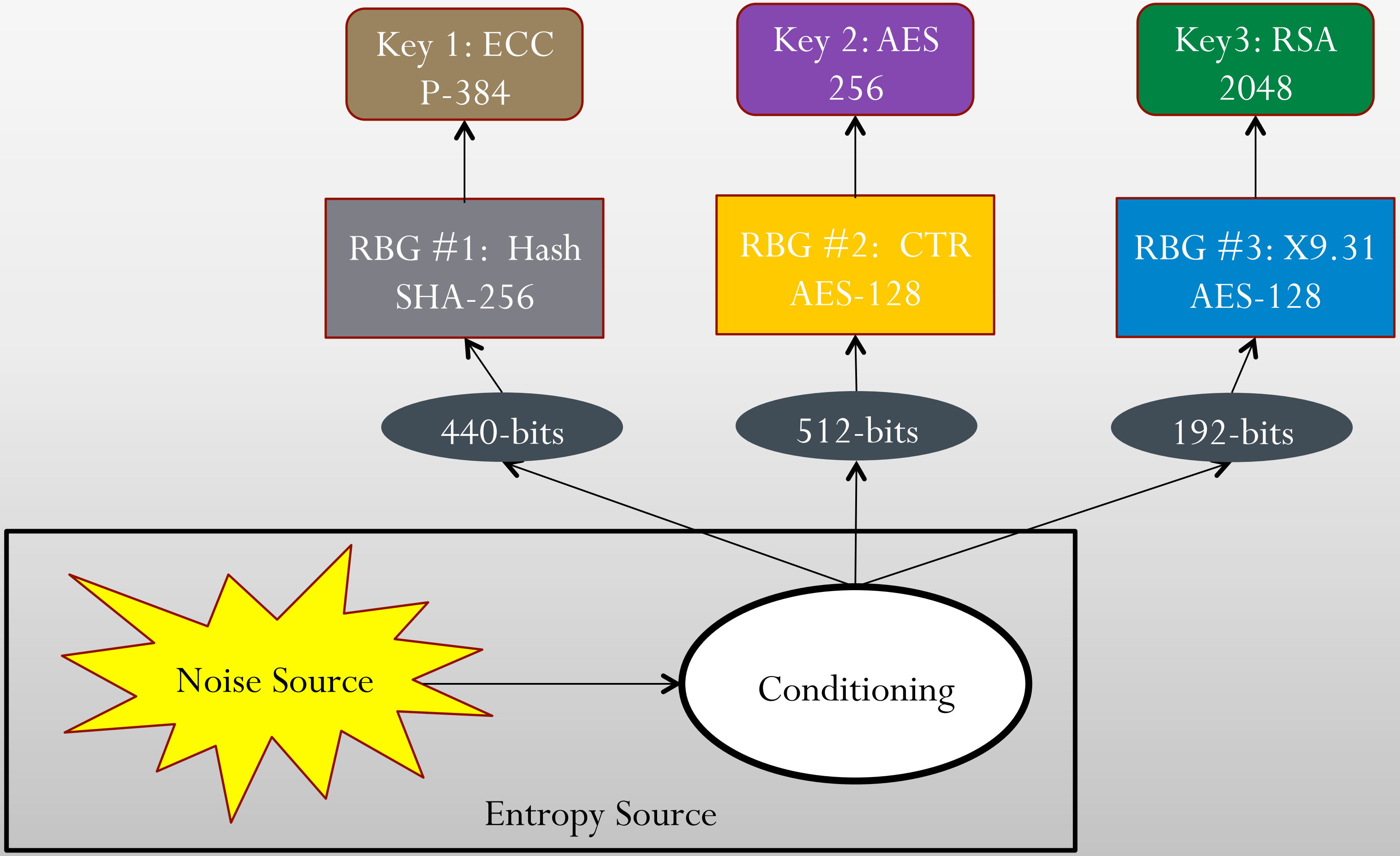
# Background (RBG Strengths)

| Standard | RBG | Bits of security | Seed Size |
|---|---|---|---|
| X9.31 | AES-128 | 128 | 256 |
| 800-90A | AES-128 CTR | 128 | 192 |
| X9.31 | AES-256 | 256 | 384 |
| 800-90A | AES-256 CTR | 256 | |
| 800-90A | SHA-256 Hash | 256 | |
| 800-90A | SHA-256 HMAC | 256 | |

# Background (Key Strengths)

| Bits of security | Symmetric key algs | DSA/DH | RSA | ECC |
|---|---|---|---|---|
| 112 | 3DES-192 | $L = 2048$<br>$N = 224$ | 2048 | 224 |
| 128 | AES-128 | $L = 3072$<br>$N = 256$ | 3072 | 256 |
| 192 | AES-192 | $L = 7680$<br>$N = 384$ | 7680 | 384 |
| 256 | AES-256 | $L = 15360$<br>$N = 512$ | 15360 | 512+ |

# Design Problems (1)



Key 1: ECC P-384

Key 2: AES 256

Key3: RSA 2048

RBG #1: Hash SHA-256

RBG #2: CTR AES-128

RBG #3: X9.31 AES-128

440-bits

512-bits

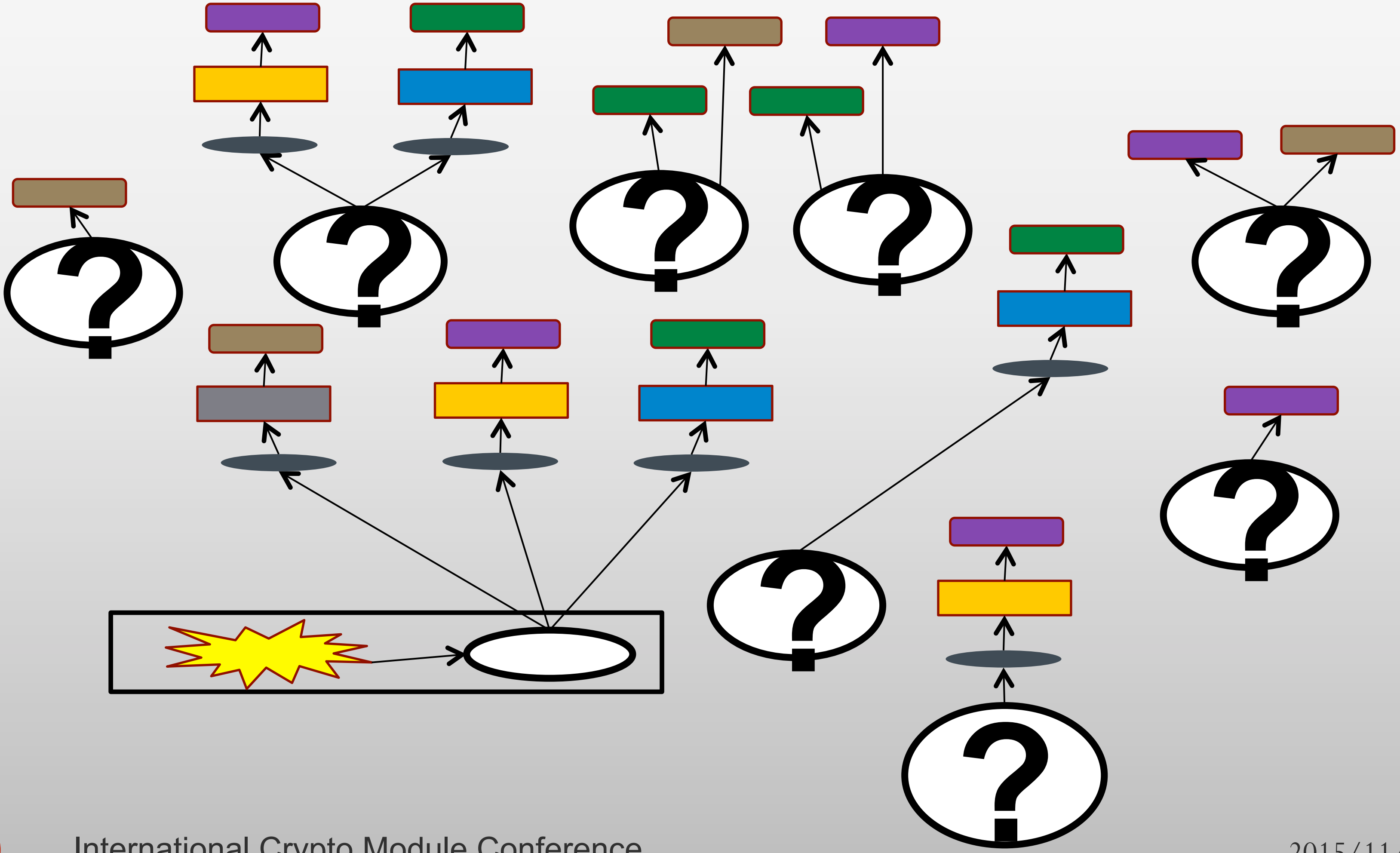
192-bits

Noise Source

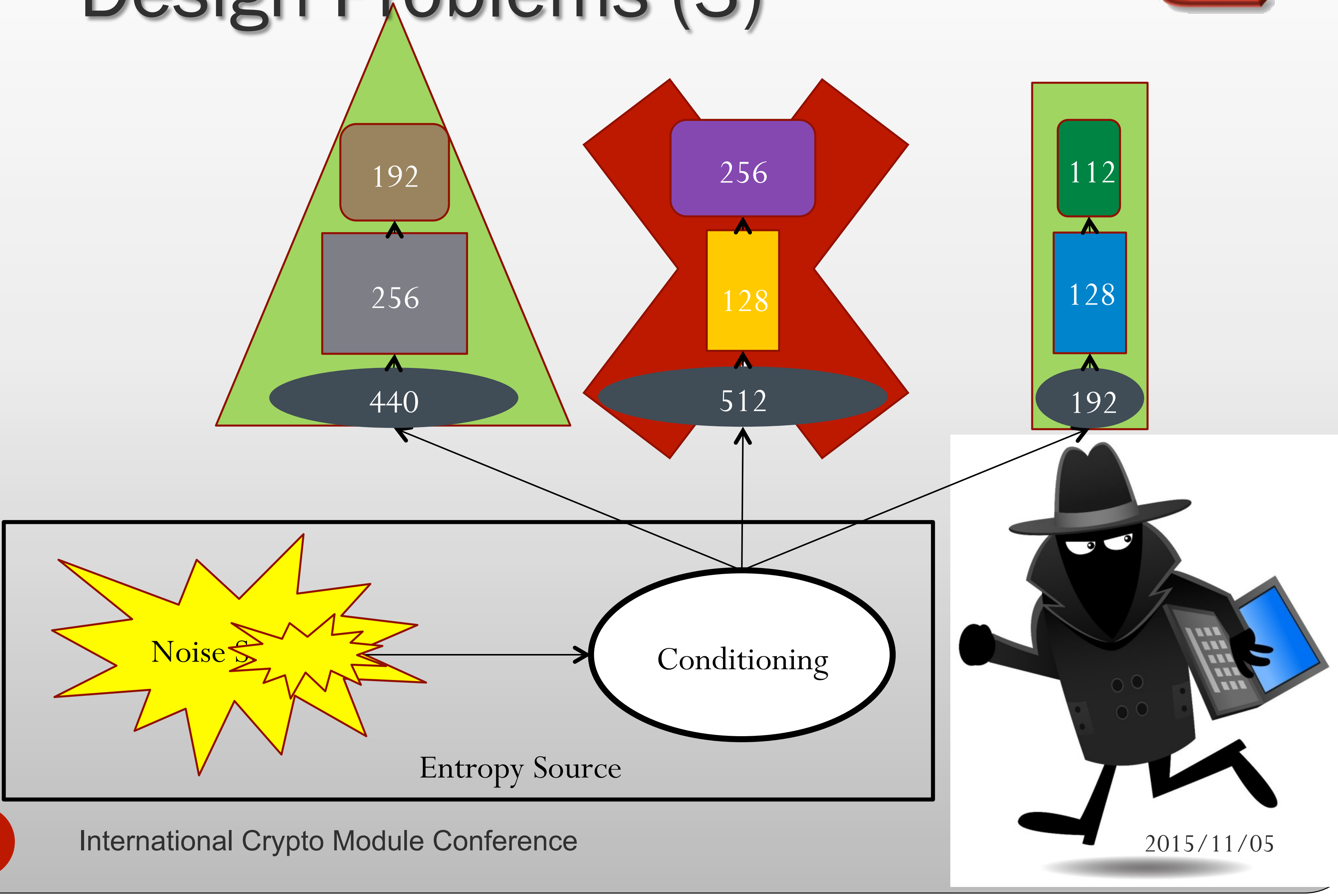
Conditioning

Entropy Source

# Design Problems (2)



# Design Problems (3)



192
256
440

256
128
512

112
128
192

Noise S

Conditioning

Entropy Source

# Design Problems (4)

- Hardware sources are generally fine
  - But it's often impossible to obtain raw samples for testing
- Software based mechanisms
  - Sample a high-frequency clock,
  - Do some operation (sleep 100 µsecs or execute code loop)
  - Sample clock again, and take difference.
- Statistical testing on SW source raw samples is extremely important

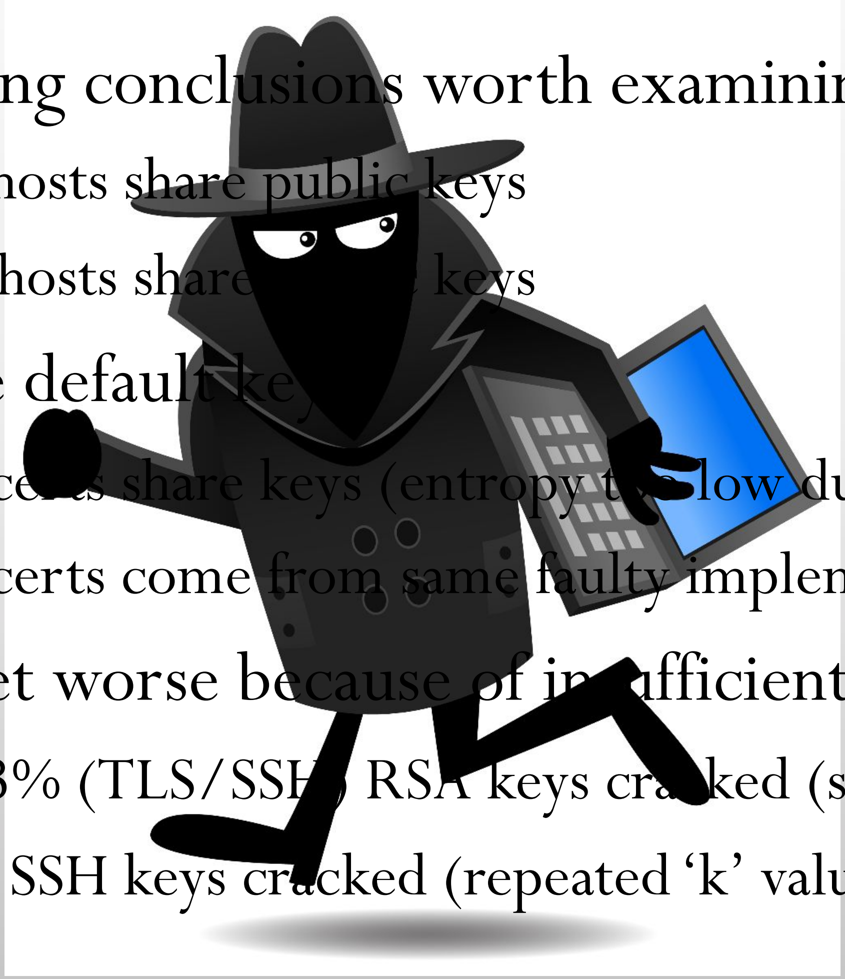
# Low Entropy Symmetric Keys

- Strong AES key (DRBG seeded w/256-bits of entropy)
  - `0xC1459F958ADB58B6CBEB54E373F52`
  - `0xEB4B7FFC4C137288FBB3F573B12E7`
- 2nd AES key (DRBG seeded w/only 40-bits of entropy)
  - `0x435F57D964A89F3853CBC98C28D8B`
  - `0xED41C9E8F9FB88171 55B489A14E89`
- Can you spot the weakness in the 2nd key?
- Don’t worry, neither can an attacker…

# Public Entropy Vulnerabilities

- 1995 Goldberg and Wagner Netscape SSL PRNG
  - PRNG (secs, usecs, pid, ppid)
- 2008 Debian OpenSSL Seeding Bug
  - Seed = PID (maximum value 32,768/2^15 )
- 2008 Karsten Nohl - weak Mifare RNG
  - 16-bit RNG depended on read time
- 2012 Lenstra et al. "Ron was Wrong, Whit is Right"
- 2012 Heninger et al. "Mining Your Ps and Qs"
  - Most comprehensive analysis of TLS/SSH public keys

# Mining Your Ps and Qs



- Some shocking conclusions worth examining
  - 5.57% TLS hosts share public keys
  - 9.60% SSH hosts share public keys
- Let's remove default keys
  - 0.75% TLS certs share keys (entropy too low during key gen)
  - 1.70% TLS certs come from same faulty implementation
- But things get worse because of insufficient entropy
  - 0.50%/0.03% (TLS/SSH) RSA keys cracked (shared primes
  - 1.03% DSA SSH keys cracked (repeated ‘k’ values)

# Mining Your Ps and Qs (2)

- But let's examine more closely
  - The authors conjecture that the use of /dev/urandom and the "Boot-time entropy hole" are to blame.
  - They continue examining OpenSSL's method of RSA key generation as an explanation for the factorable RSA keys.
  - They posit Dropbear SSH seeding with insufficient entropy /dev/urandom
- These problems are less concerning upon inspection
- Again, for symmetric keys generation, these problems don't exist

# Recommendations

- Avoid foolish mistakes
  - Do not use `/dev/`**`u`**`random` (use `/dev/random` instead)
  - Use simple post-processing/conditioning (or none)
  - Watch for entropy bottlenecks
  - Mix entropy sources appropriately (XOR or pool Hashing) and account for the weighted contribution of each source
  - Scrutinize entropy at a cold boot/first boot
  - Do not automatically generate key pairs during boot
  - Scrutinize software noise sources (hardware noise sources are generally good)

# Recommendations (2)

- Oversample whenever possible, be conservative
- Accept tradeoffs to increase security
- Test software entropy quality on all hardware models
- Introduce device unique data (e.g., seed at factory)
- Use hardware noise or evaluated sources if available

- Catastrophic failures are rare
  - 40-bits of entropy likely to be indistinguishable from 256-bits once fed through an appropriate DRBG

# Questions?

**Contacts:**

- Ed Morris
  - EdMorris@gossamersec.com
- Khai Van
  - KhaiVan@gossamersec.com

www.gossamersec.com

www.facebook.com/gossamersec

@gossamersec