Lecture 7 – Randomness, entropy, TRNG/PRNGs, stream ciphers

TEK4500

05.10.2022

Håkon Jacobsen

hakon.jacobsen@its.uio.no

Midterm exam

- Available: Wednesday 5. October
- Due: **two weeks** later (Thursday 20. October, 23:59)
- **Take-home** exam
- **Individual:** collaboration is *not* allowed
- **Mandatory:** need to pass in order to be eligible for the exam
- All sources allowed (save for explicitly searching for the solution)
- Submission: Canvas (file type = PDF; strongly prefer if you use provided LaTex template)
- Start early! The assignment may be more challenging than you expect

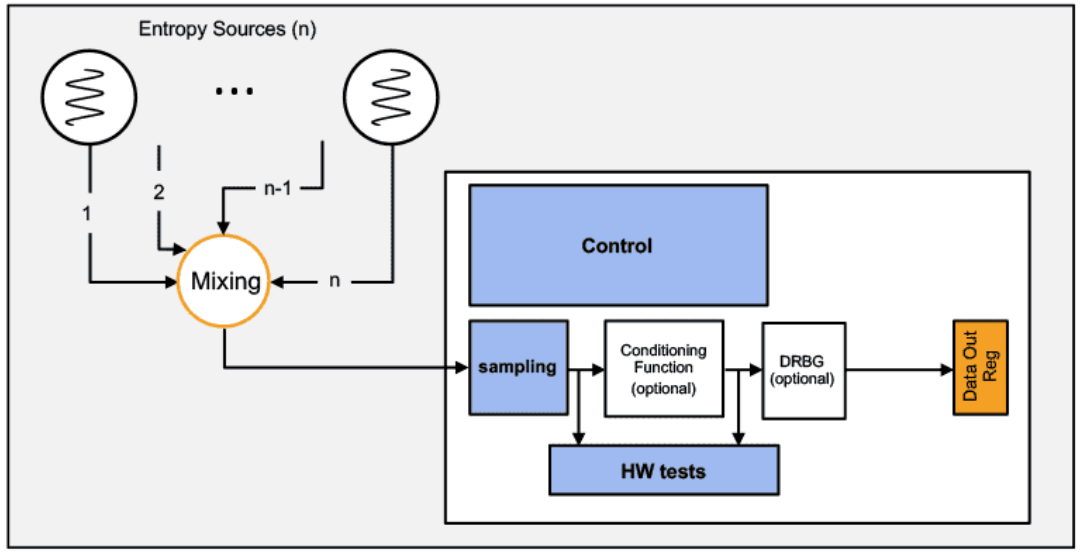
Outline

- How to generate (lots) of randomness?
- PRNGs
  - Stream ciphers
- Randomness – what is it?
- TRNGs
- Key derivation

Random generators

- Common design:
  - TRNG generates random bits
  - Random bits are compressed to short seed
  - PRNG expands seed to “infinite” length

- Examples:
  - /dev/urandom
  - CryptGenRandom
  - Intel RDRAND

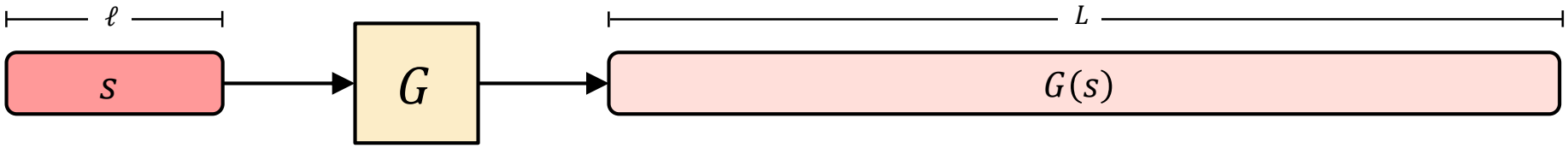


Pseudorandom generators (PRG) – syntax

Have: a short string s in $\{0,1\}^{\ell}$ (**uniform** and **independently** distributed)

Want: a *long* string S in $\{0,1\}^{L}$ (**uniform** and **independently** distributed)

Solution: a **pseudorandom generator (PRG)**, i.e. a function $G : \{0,1\}^{\ell} \to \{0,1\}^{L}$

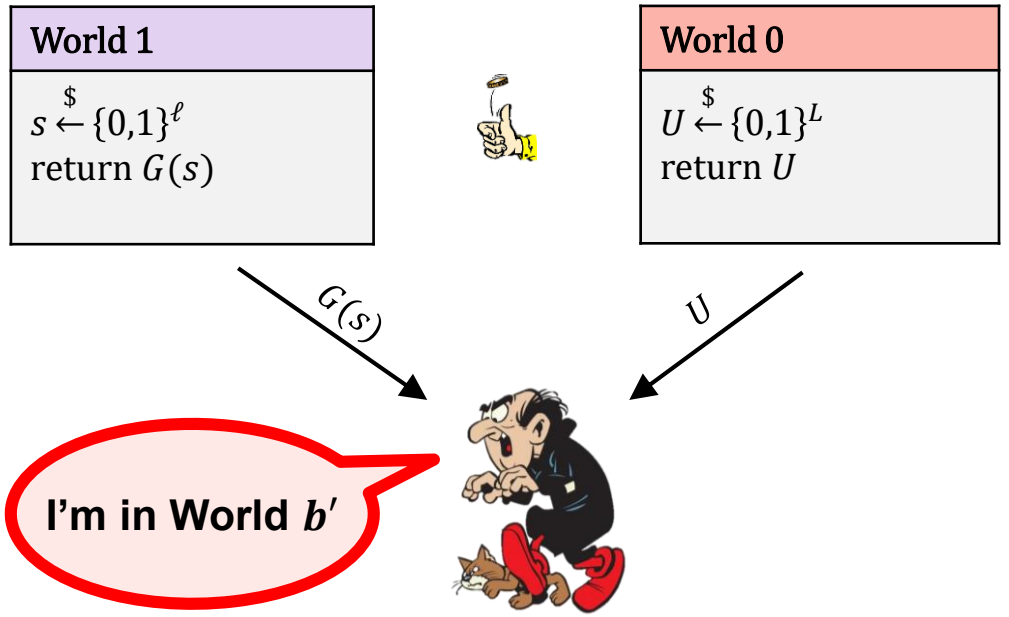


- Expansion: $L \gg \ell$
- Pseudorandomness: $G(s)$ should look like a truly random string $U \xleftarrow{\$} \{0,1\}^{L}$

PRNG – security definition

$\mathbf{Exp}_G^{\mathrm{prg}}(A)$

1. $b \xleftarrow{\$} \{0,1\}$
2. $s \xleftarrow{\$} \{0,1\}^{\ell}$
3. $U_1 \leftarrow G(s)$
4. $U_0 \xleftarrow{\$} \{0,1\}^{L}$
5. $b' \leftarrow A(U_b)$
6. **return** $b' \stackrel{?}{=} b$



Definition: The **PRNG advantage** of an adversary A is

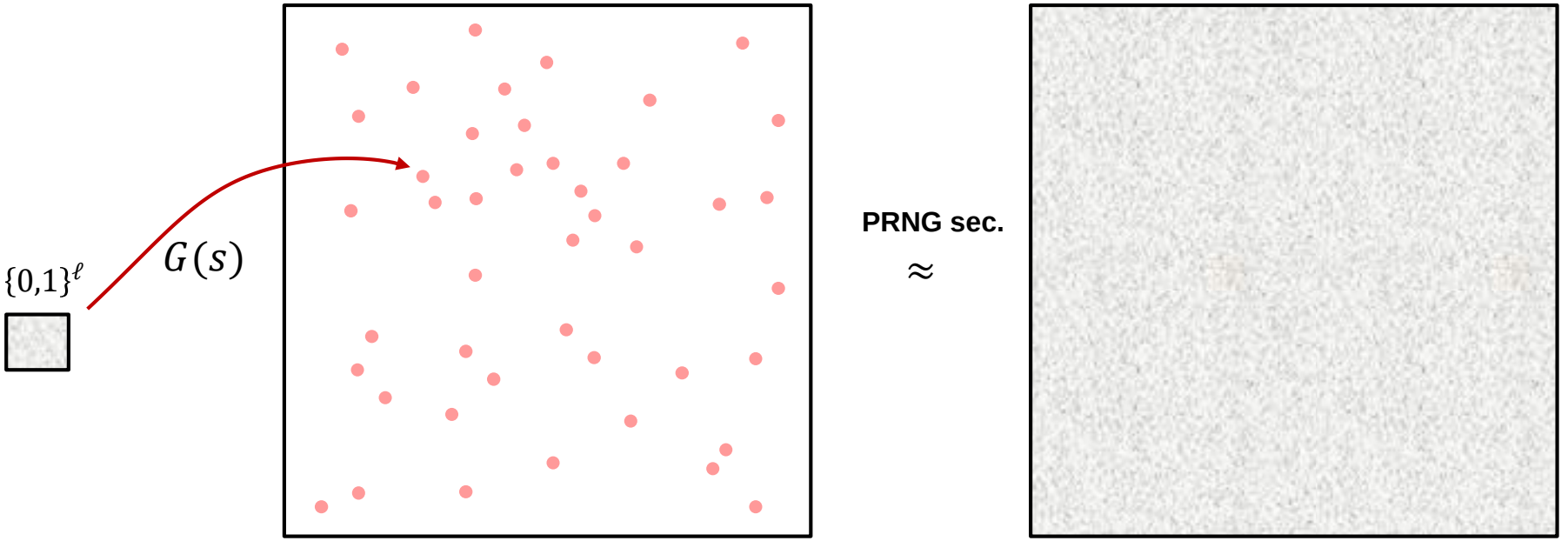
$$\mathbf{Adv}_G^{\mathrm{prg}}(A) = \left|2 \cdot \Pr\left[\mathbf{Exp}_G^{\mathrm{prg}}(A) \Rightarrow \mathrm{true}\right] - 1\right|$$

$$2 \cdot \Pr[b' = b] - 1 = \Pr[\, b' = 1 \mid b = 1\,] + \Pr[\, b' = 0 \mid b = 0\,] - 1$$

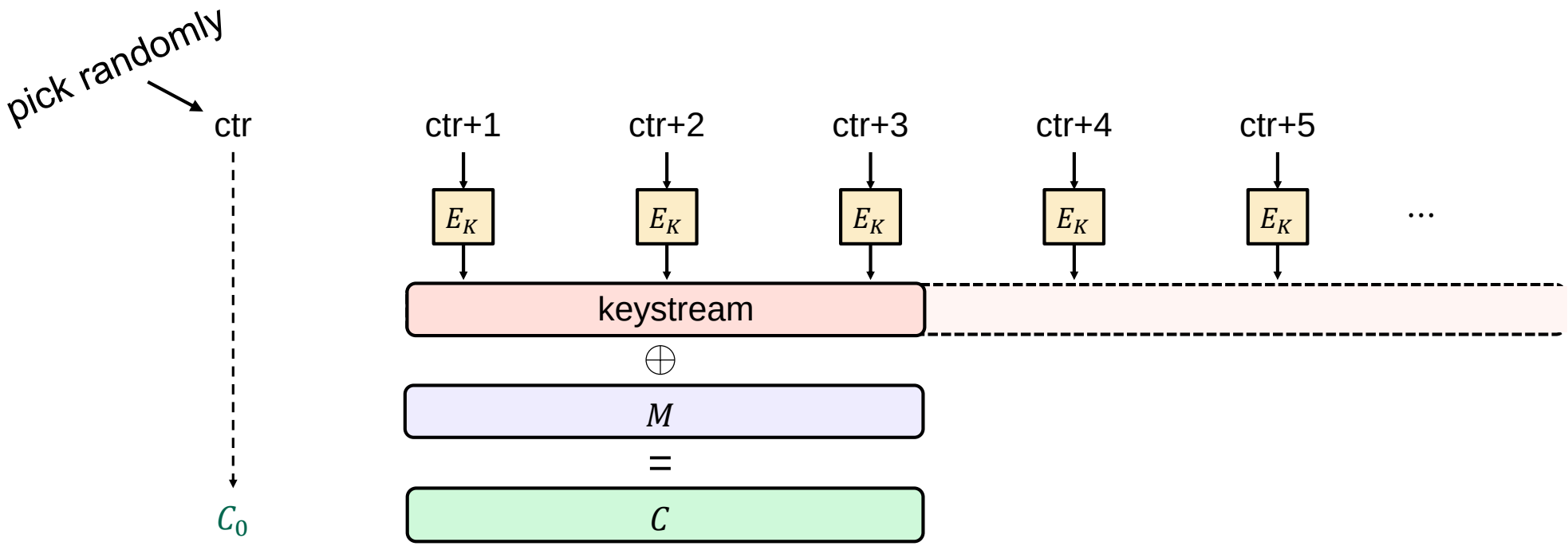
$$= \Pr[\, b' = 1 \mid b = 1\,] - \Pr[\, b' = 1 \mid b = 0\,]$$

$$= \Pr\left[A\left(G(s)\right) \Rightarrow 1\right] - \Pr[A(U) \Rightarrow 1]$$

Pseudorandomness

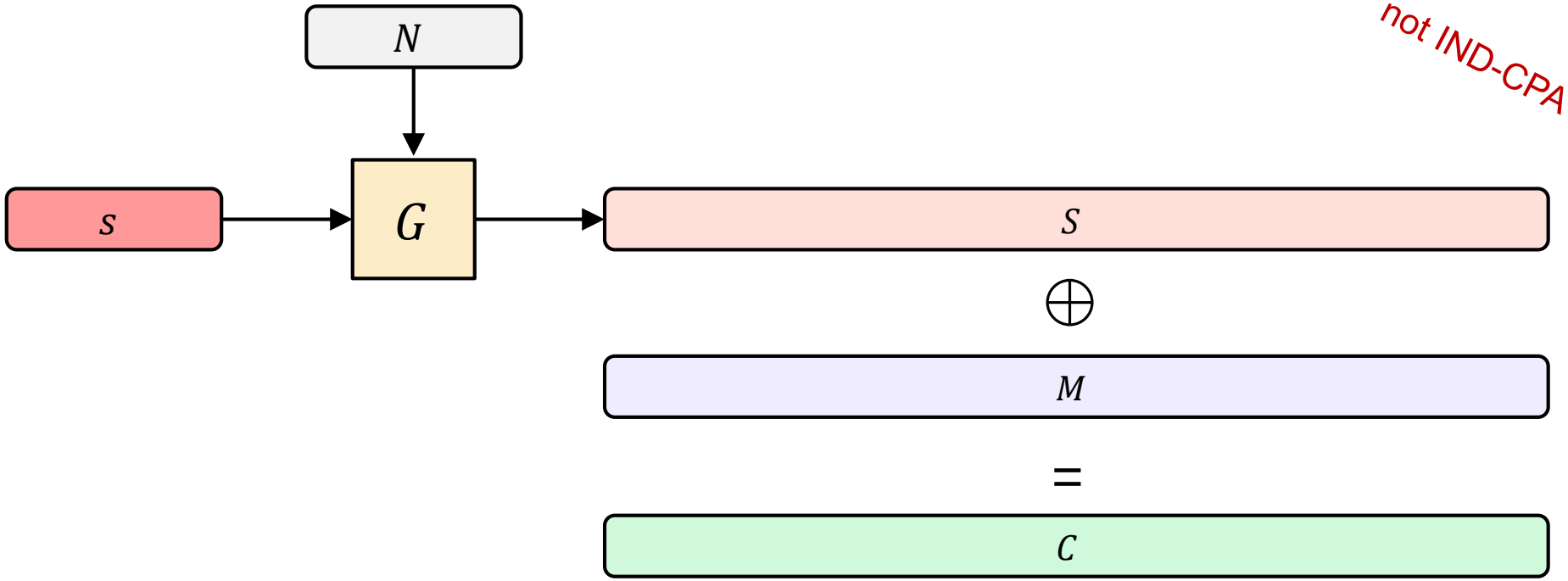


Creating PRNGs – CTR-mode



Stream ciphers

Security requirement: For all N the function $G(\cdot, N)$ is a secure PRG

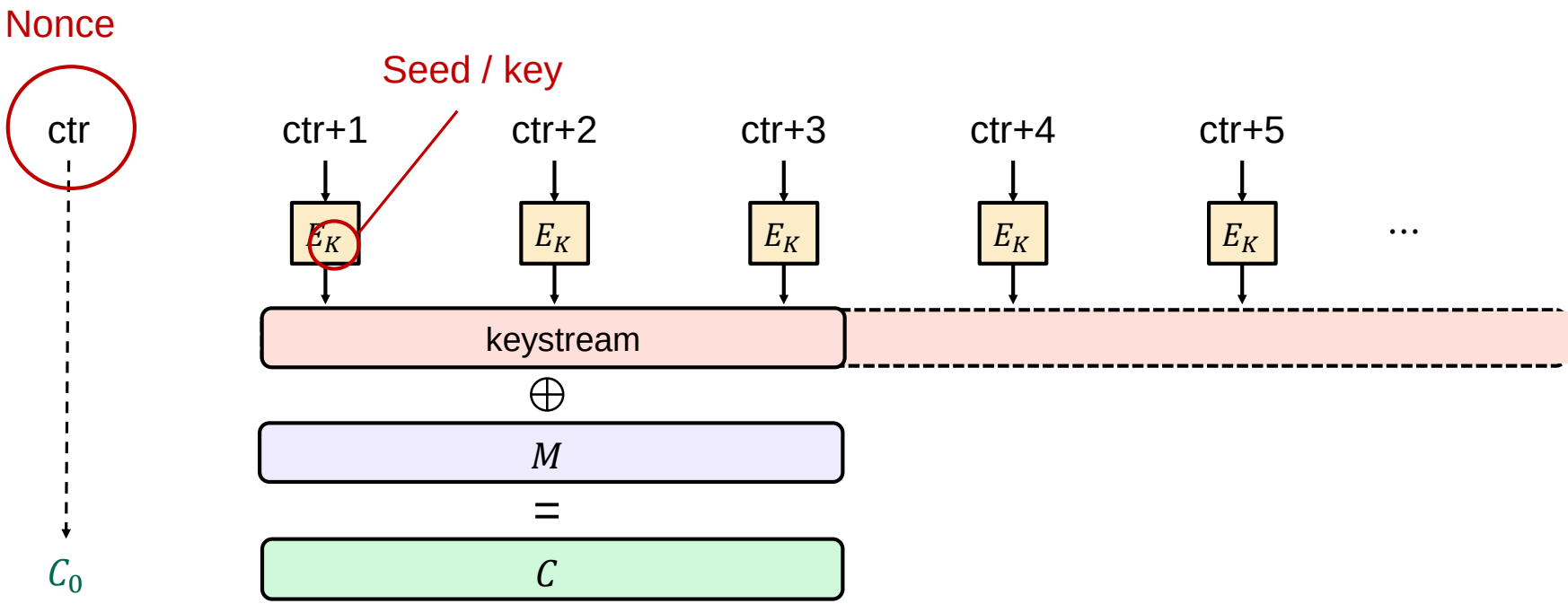


not IND-CPA secure!

$\text{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{M} \rightarrow \mathcal{C}$

$\text{Enc}_K^N(M) = N \,||\, G(K, N) \oplus M$ ← **Stream cipher**

Stream ciphers – CTR-mode



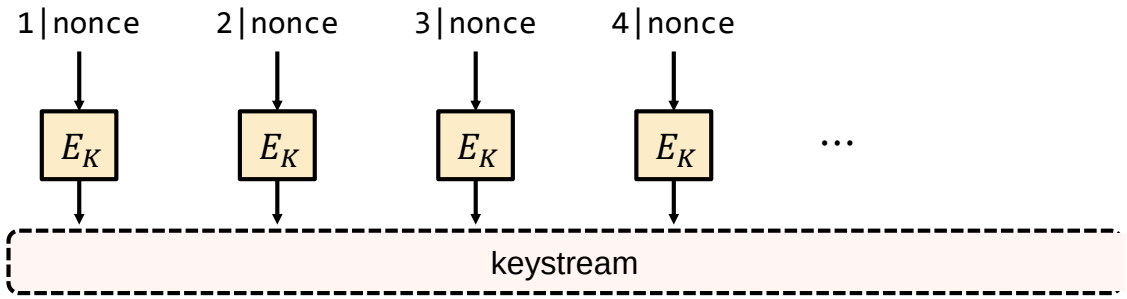
ChaCha20

$E : \{0,1\}^{256} \times \{0,1\}^{128} \rightarrow \{0,1\}^{512}$

```
Algorithm QR(a, b, c , d)

1.  a += b;   d ^= a;   d <<<= 16;
2.  c += d;   b ^= c;   b <<<= 12;
3.  a += b;   d ^= a;   d <<<=  8;
4.  c += d;   b ^= c;   b <<<=  7;
```

```
Algorithm E(key, ctr | nonce)


1.    state ← const | const | const | const
2.            key   | key   | key   | key
3.            key   | key   | key   | key
4.            ctr   | nonce | nonce | nonce
5.
6.    s ← state
7.
8.    for i = 1 to 10 do
9.      QR(s[0],  s[4],  s[8],  s[12])
10.     QR(s[1],  s[5],  s[9],  s[13])
11.     QR(s[2],  s[6],  s[10], s[14])
12.     QR(s[3],  s[7],  s[11], s[15])
13.     QR(s[0],  s[5],  s[10], s[15])
14.     QR(s[1],  s[6],  s[11], s[12])
15.     QR(s[2],  s[7],  s[8],  s[13])
16.     QR(s[3],  s[4],  s[9],  s[14])
17.   end for
18.
19.   state += s
20.   return state
```

ChaCha20:



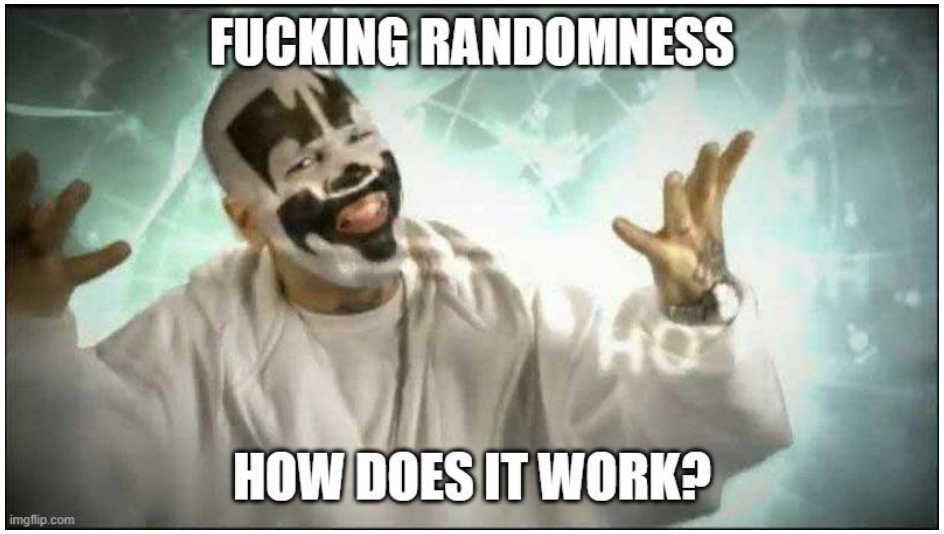
$$s = \begin{matrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \\ 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{matrix}$$

ChaCha20

- Extremely simple
  - only ARX operations (add-rotate-xor)
- Very fast in SW
  - no HW support required
  - faster than AES(-CTR) without AES-NI instructions
- Constant time – no tables
- No key-setup, no subkeys
- Often combined with the MAC Poly1305 to create an AEAD
- Designed by Daniel J. Bernstein

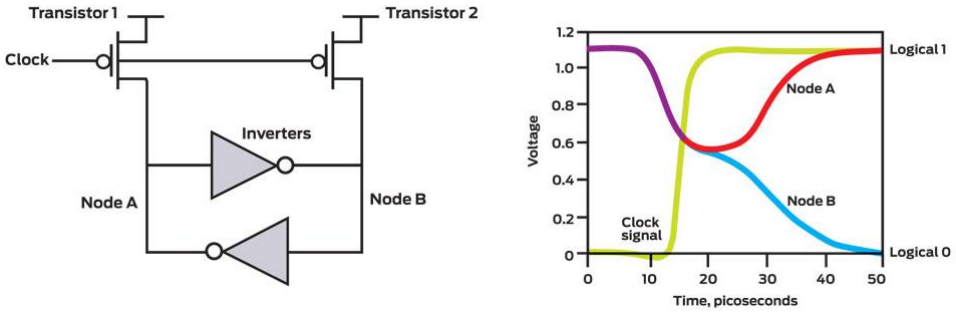
```
Algorithm E(key, ctr | nonce)


1.    state ← const | const | const | const
2.            key   | key   | key   | key
3.            key   | key   | key   | key
4.            ctr   | nonce | nonce | nonce
5.
6.    s ← state
7.
8.    for i = 1 to 10 do
9.      QR(s[0],  s[4],  s[8],  s[12])
10.     QR(s[1],  s[5],  s[9],  s[13])
11.     QR(s[2],  s[6],  s[10], s[14])
12.     QR(s[3],  s[7],  s[11], s[15])
13.     QR(s[0],  s[5],  s[10], s[15])
14.     QR(s[1],  s[6],  s[11], s[12])
15.     QR(s[2],  s[7],  s[8],  s[13])
16.     QR(s[3],  s[4],  s[9],  s[14])
17.   end for
18.
19.   state += s
20.   return state
```

Randomness



TRNG

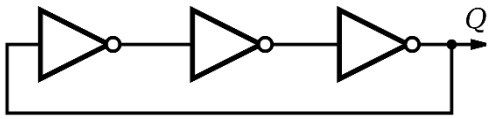
Thermal noise



Entropy sources

https://www.cloudflare.com/learning/ssl/lava-lamp-encryption/

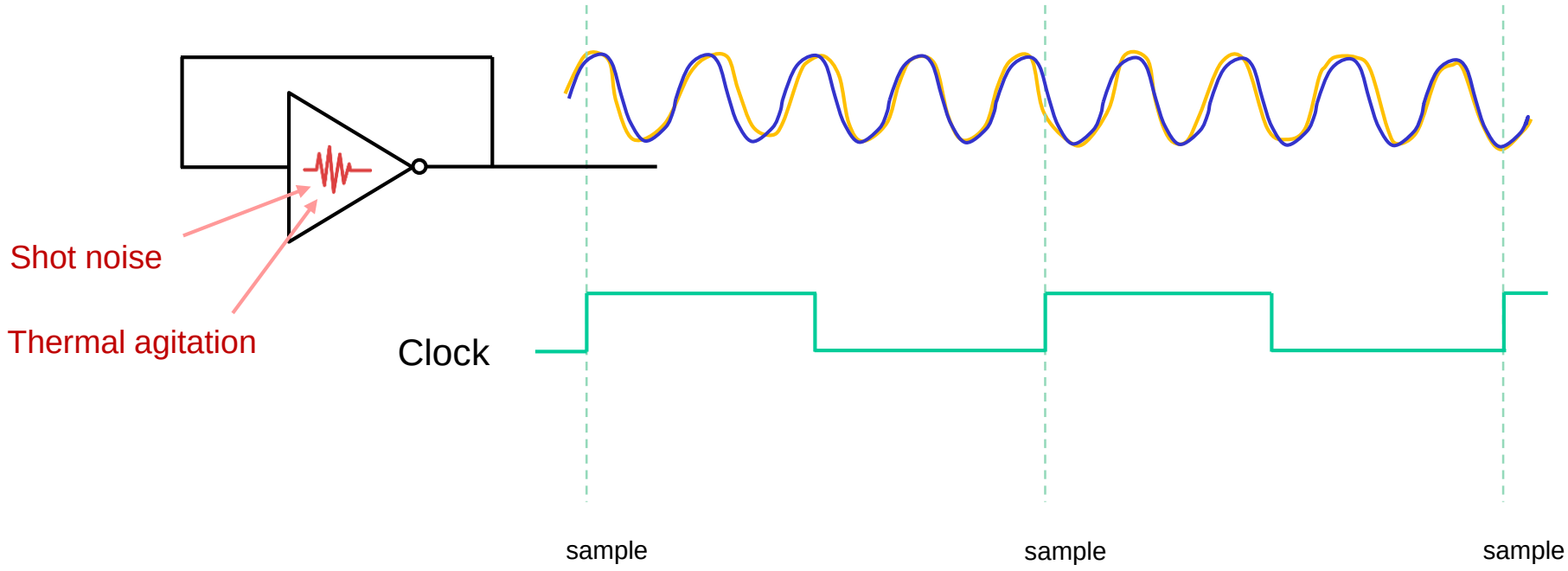
Ring oscillators



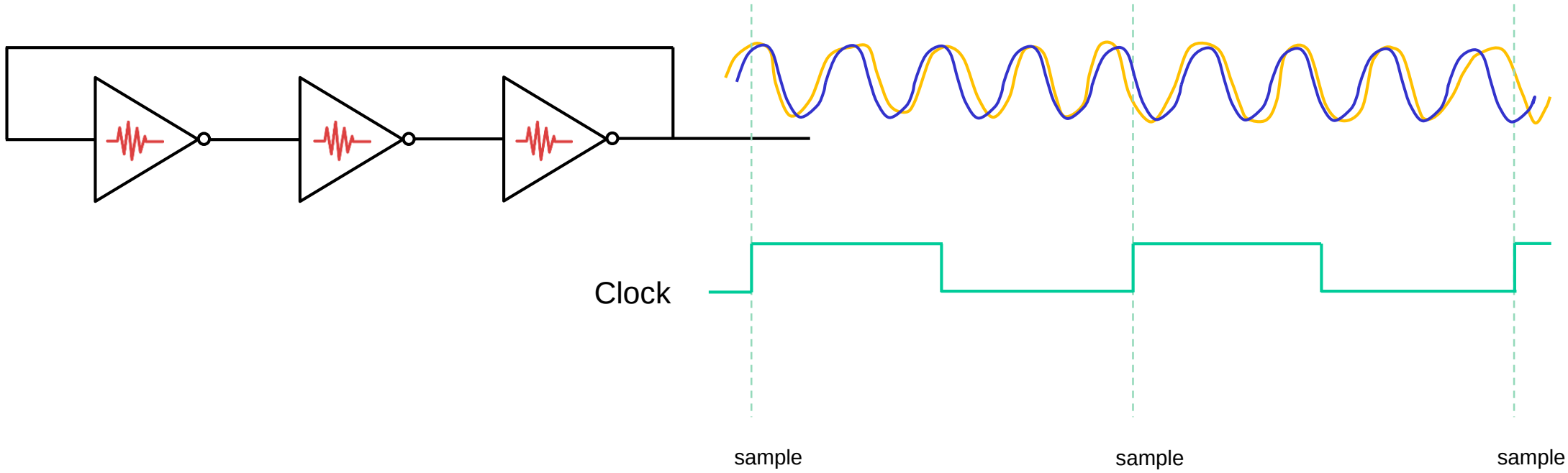
Quantum magic

(radioactive decay, quantum tunneling, etc…)

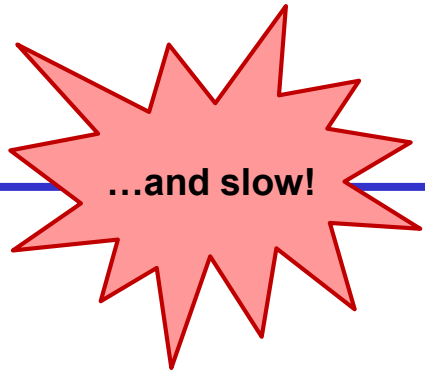
Ring-oscillators



Ring-oscillators



Problems with TRNGs



- **Biased sources**
  - E.g. bit 0 with probability 0.25 and bit 1 with probability 0.75
  - Symmetric schemes require *uniform* keys
  - De-bias (von Neumann): create *two* bits; $01 \mapsto 0$, $10 \mapsto 1$, $00/11 \mapsto \text{try again}$ (in practice: hash with SHA2-256)

- **Correlated sources**
  - Value of bit 73 may depend on bit 5
  - Symmetric schemes require *independent* keys
  - De-correlate: much more difficult! (in practice: hash with SHA2-256)

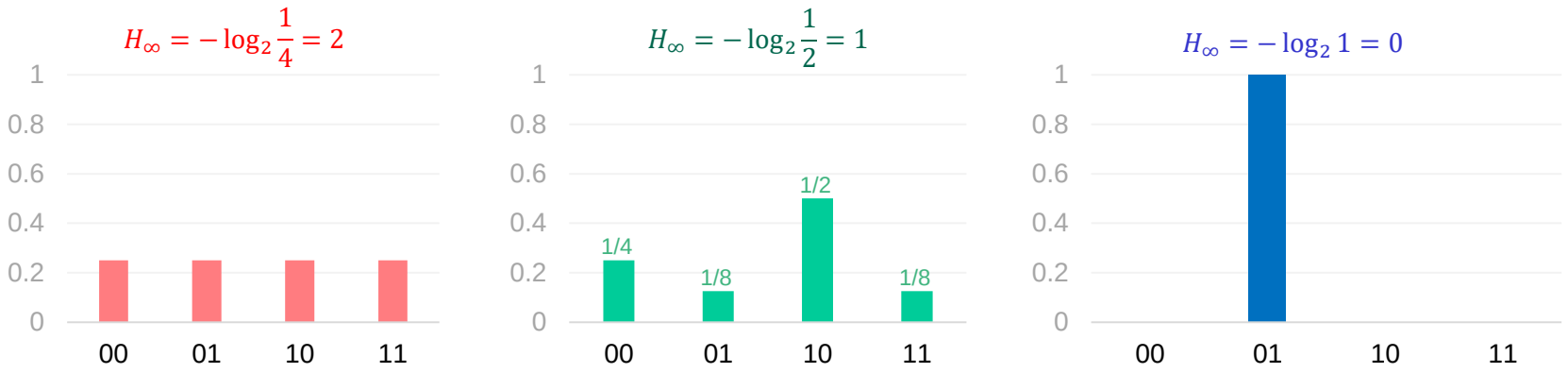
Statistical tests

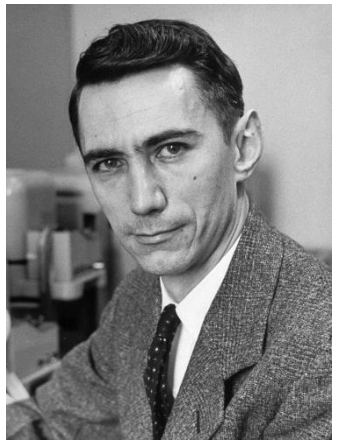
101010011101011000100100011111000010101000000001010011111

- Is this a random string? Our answer: question not valid!
- What does that even mean?
- Suggestions:
  - A random string should have roughly 50% zeros and ones (how much can you deviate?)
  - A continuous run of zeros (or ones) shouldn't be too long (how long?)
  - ≈ 25% of 2-bit substrings should be 00, 25% should be 01, …
  - ≈ 12.5% of 3-bit substrings should be 000, 12.5% should be 001, ….
  - A random string should not be compressible (related to Kolmogorov-complexity)
  - …

Entropy

- Measure of uncertainty
  - Measured in bits
  - $H_\infty = \text{min-entropy} \stackrel{\text{def}}{=} -\log_2\left(\max_x \Pr[x]\right)$
  - $\Pr[\text{best guessing strategy}] \leq 2^{-H_\infty}$

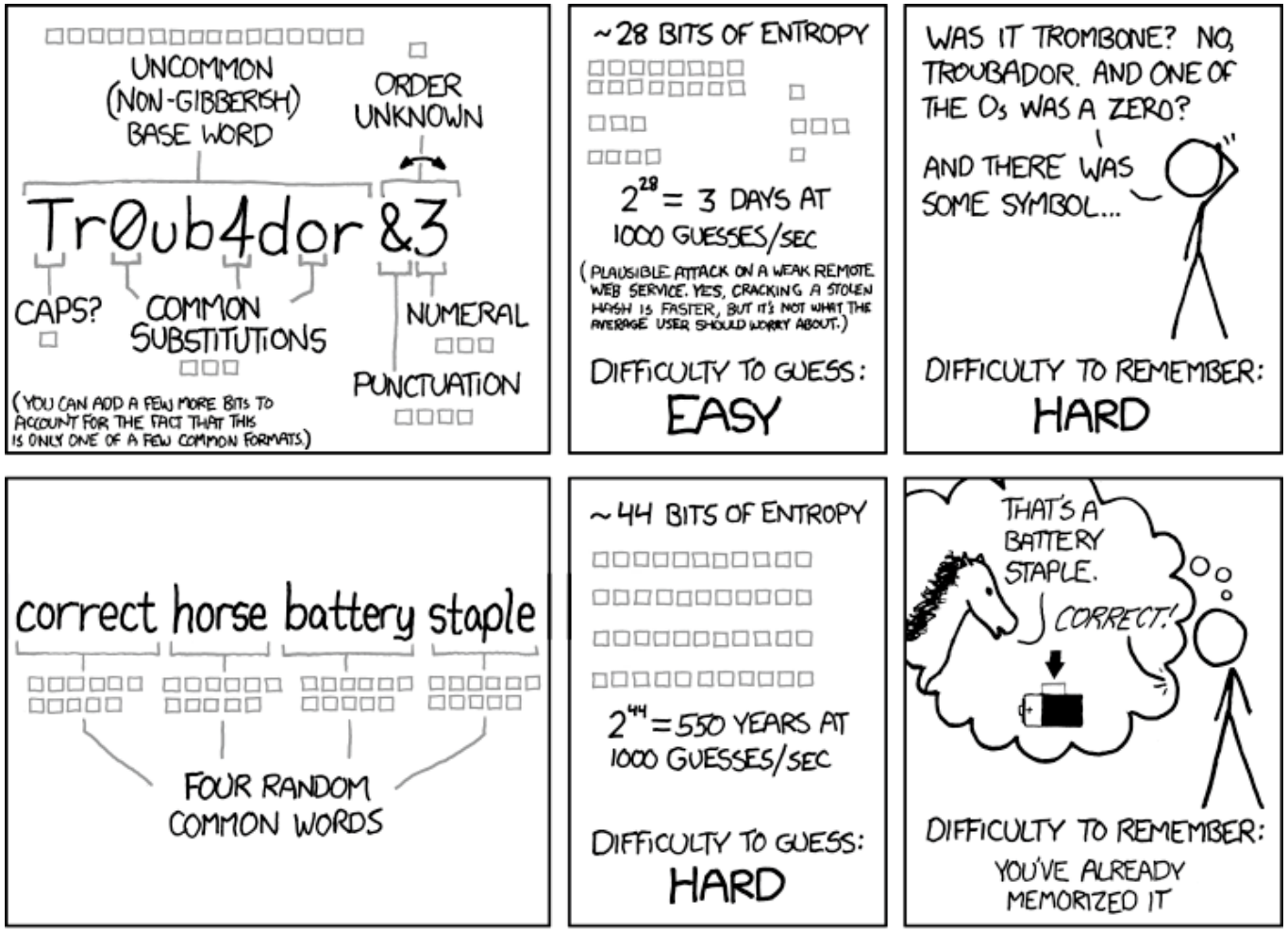




Claude Shannon

- **Examples:**
  - Fair coin: $H_\infty = 1$
  - Fair 6-sided die: $H_\infty = -\log_2 \frac{1}{6} \approx 2.58$
  - Uniform 128-bit string: $H_\infty = 128$
  - Uniform n-bit string + uniform m-bit string: $H_\infty = n + m$

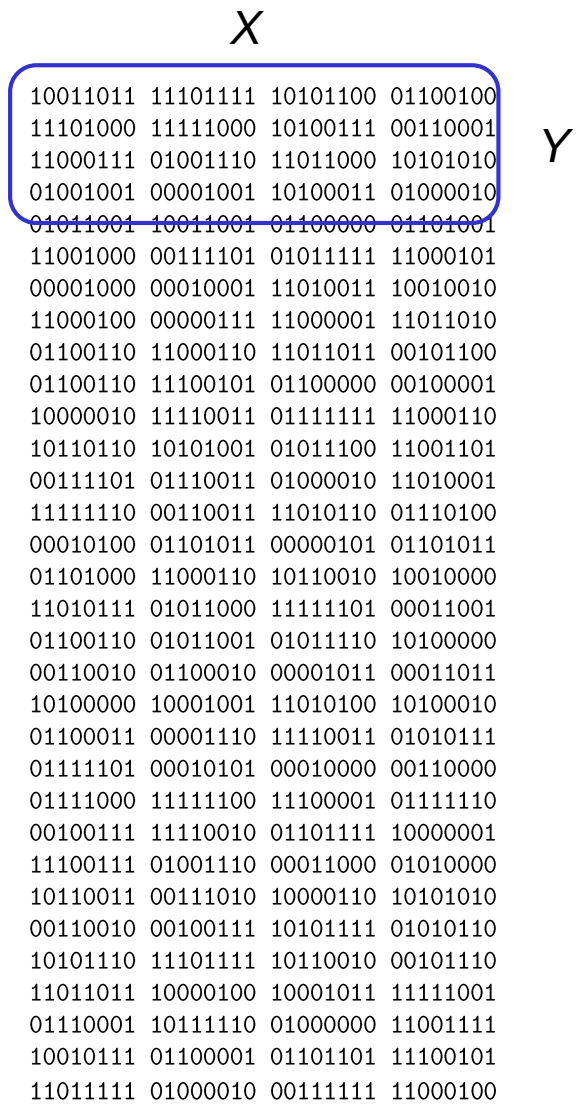
Password strength



https://xkcd.com/936

Entropy extractors

- Have a (long) string *X* with 128 bits of min-entropy
- Want 128 bits for AES key *Y*. How to choose?



Entropy extractors

- Have a (long) string *X* with 128 bits of min-entropy
- Want 128 bits for AES key *Y*. How to choose?
- Bits of *X* may be *biased* and *dependent*
- Want to "concentrate" the 128 bits of entropy in *X* into a string *Y* of only 128 bits
- Bits of *Y* should be *non-biased* and *independent*

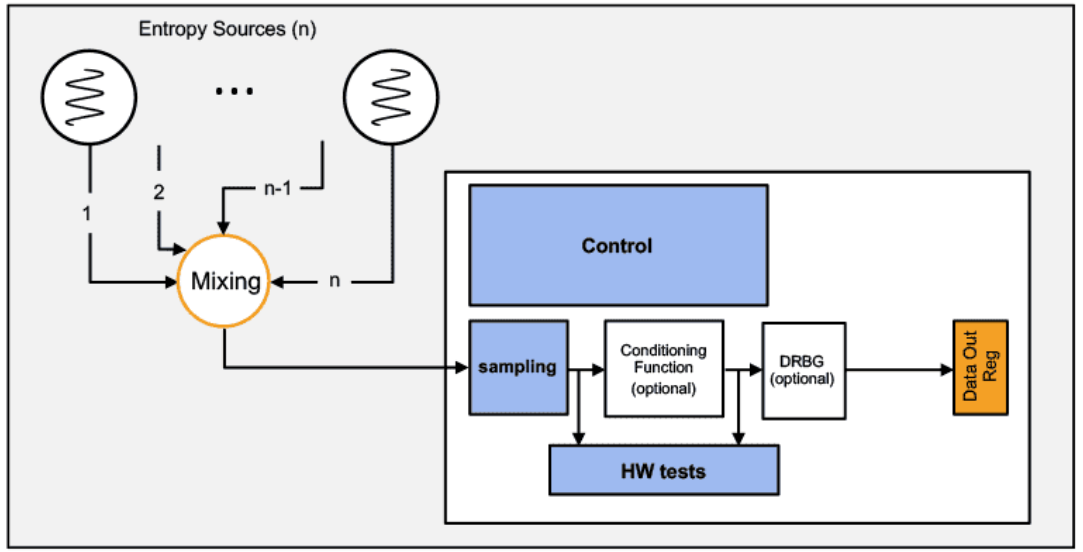
X

Y

00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
01011001 10011001 01100000 01101001
11001000 00111101 01011111 11000101
00001000 00010001 11010011 10010010
11000100 00000111 11000001 11011010
01100110 11000110 11011011 00101100
01100110 11100101 01100000 00100001
10000010 11110011 01111111 11000110
10110110 10101001 01011100 11001101
00111101 01110011 01000010 11010001
11111110 00110011 11010110 01110100
00010100 01101011 00000101 01101011
01101000 11000110 10110010 10010000
11010111 01011000 11111101 00011001
01100110 01011001 01011110 10100000
00110010 01100010 00001011 00011011
10100000 10001001 11010100 10100010
01100011 00001110 11110011 01010111
01111101 00010101 00010000 00110000
01111000 11111100 11100001 01111110
00100111 11110010 01101111 10000001
11100111 01001110 00011000 01010000
10110011 00111010 10000110 10101010
00110010 00100111 10101111 01010110
10101110 11101111 10110010 00101110
11011011 10000100 10001011 11111001
01110001 10111110 01000000 11001111
10010111 01100001 01101101 11100101
11011111 01000010 00111111 11000100

Random generators

- Common design:
  - TRNG generates random bits
  - Random bits are compressed to short seed
  - PRNG expands seed to “infinite” length



- Examples:
  - /dev/urandom
  - CryptGenRandom
  - Intel RDRAND

- Debian OpenSSL RNG bug
  - `// MD_Update(&m,buf,j);`
  - Only 32,767 possibilities for seed ≈ 15 bits of entropy

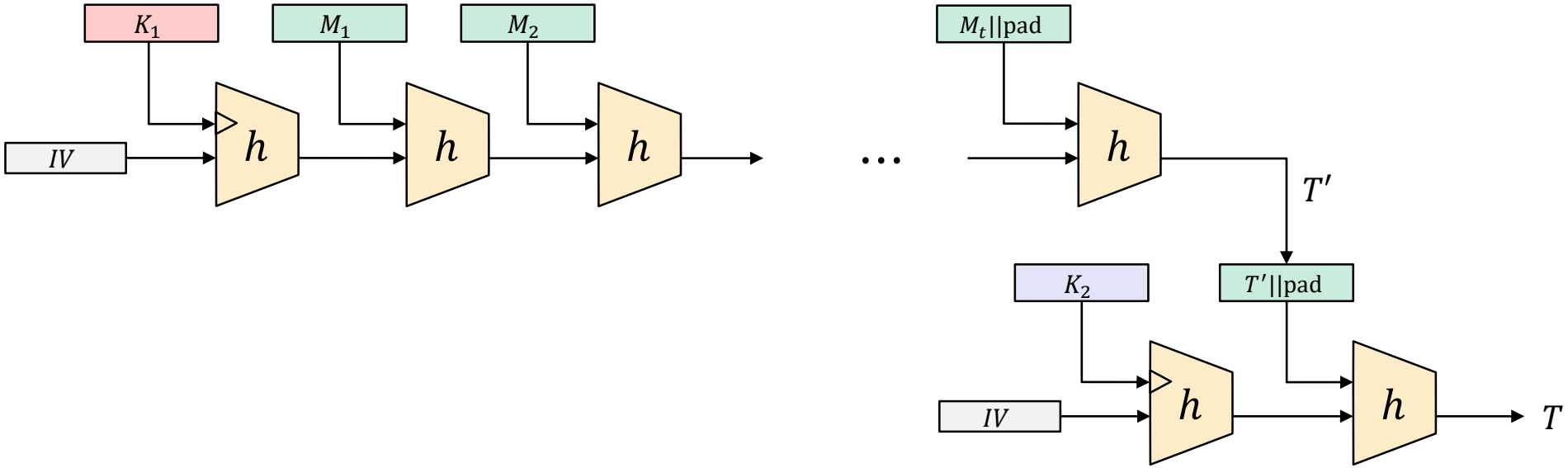
Key derivation functions (KDF)

- Applications typically need many keys
  - Traffic encryption keys (send + receive)
  - Traffic MAC keys (send + receive)
  - Data storage keys
  - …

- **KDF:** transform a source of keying material into one or more pseudorandom keys

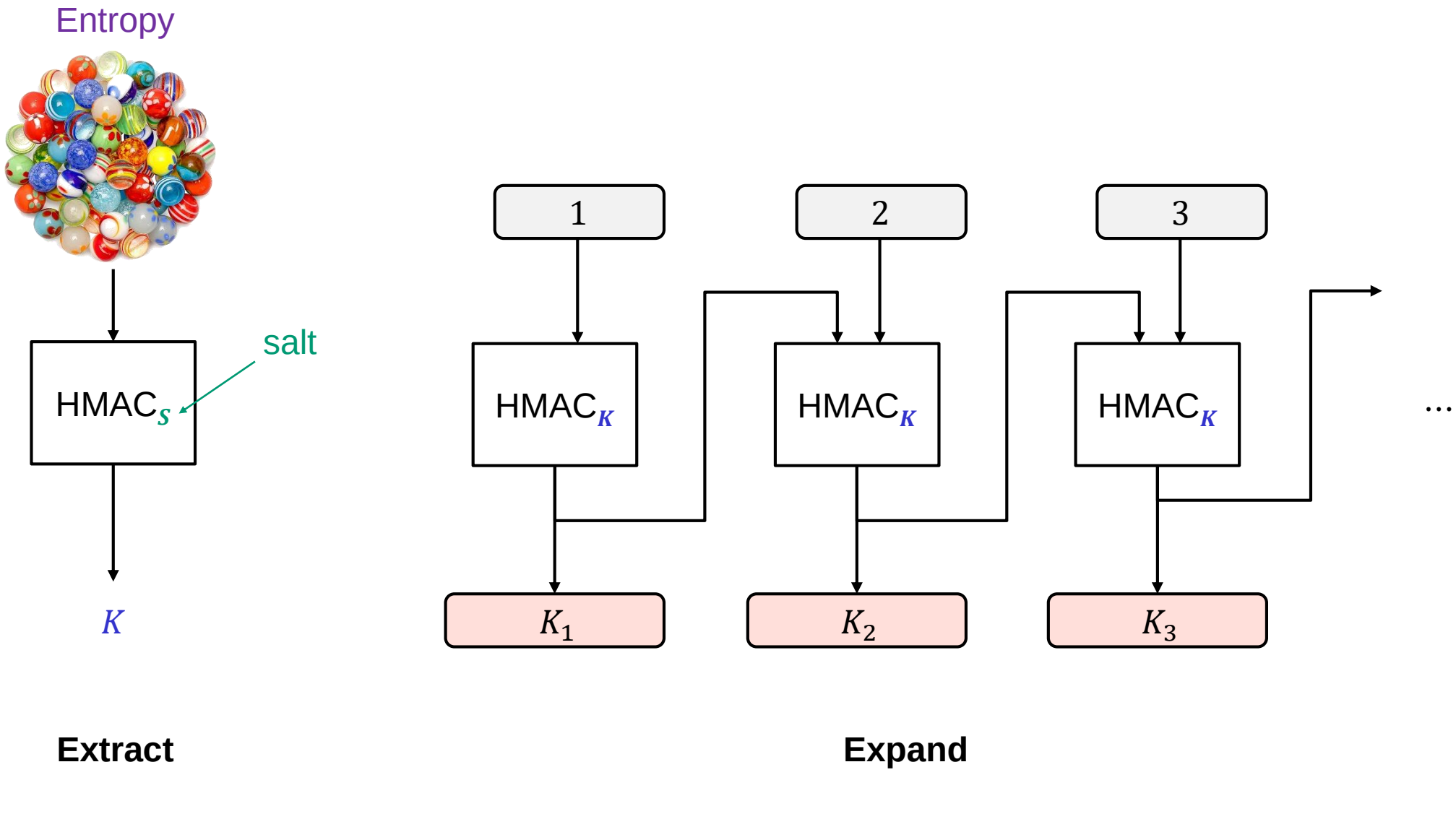
- Common paradigm: **extract-then-expand**
  - Extract: compress high-entropy input key material into short uniform key with high entropy
  - Expand: expand short key into many using a PRNG

Recap: HMAC

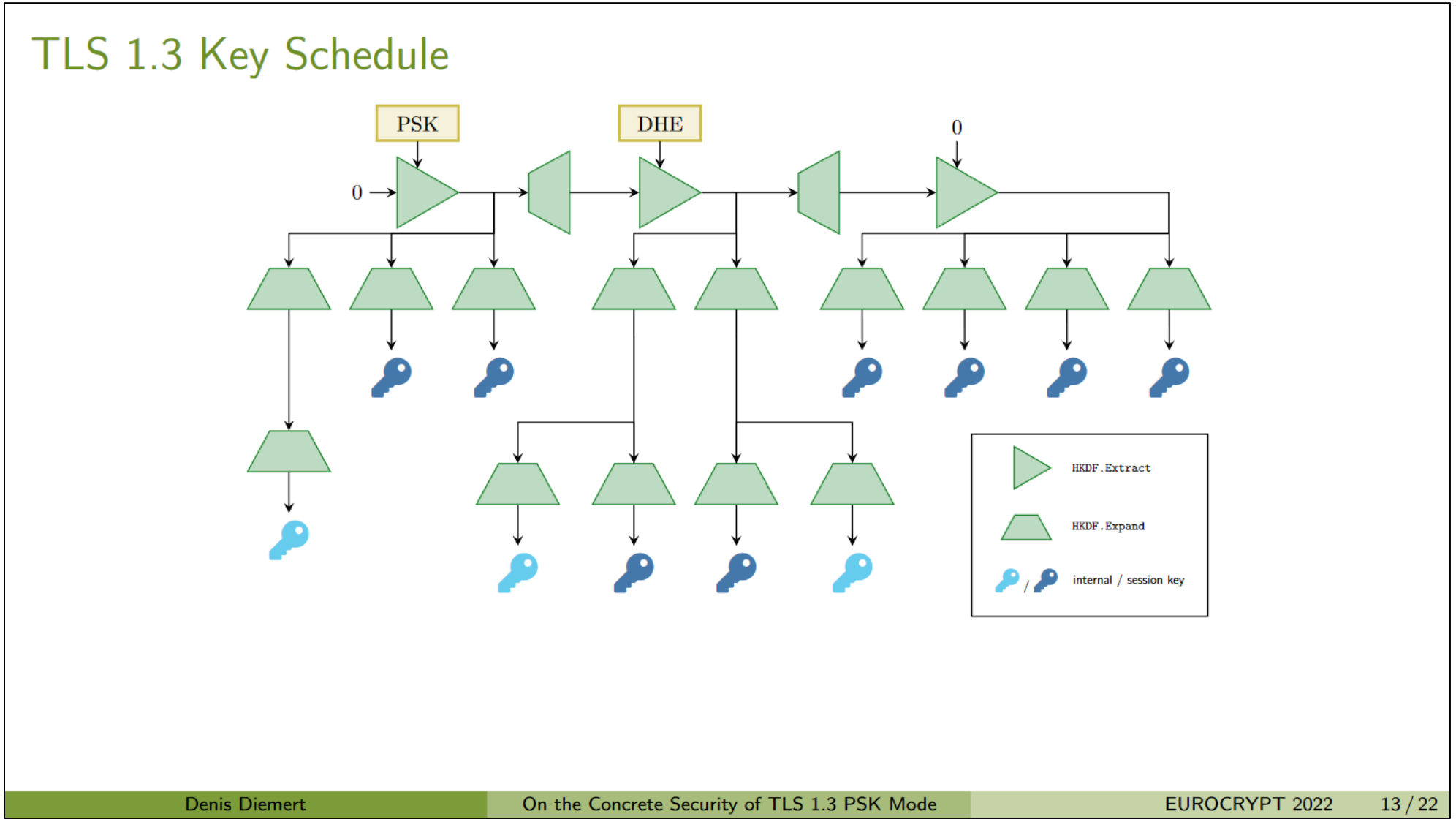


$$\mathrm{HMAC}_K(M) = H\big(\, K \oplus \mathrm{opad} \;\|\; H(K \oplus \mathrm{ipad} \,\|\, M)\big)$$

HKDF – HMAC-based KDF



Key derivations in practice



LTE key derivation

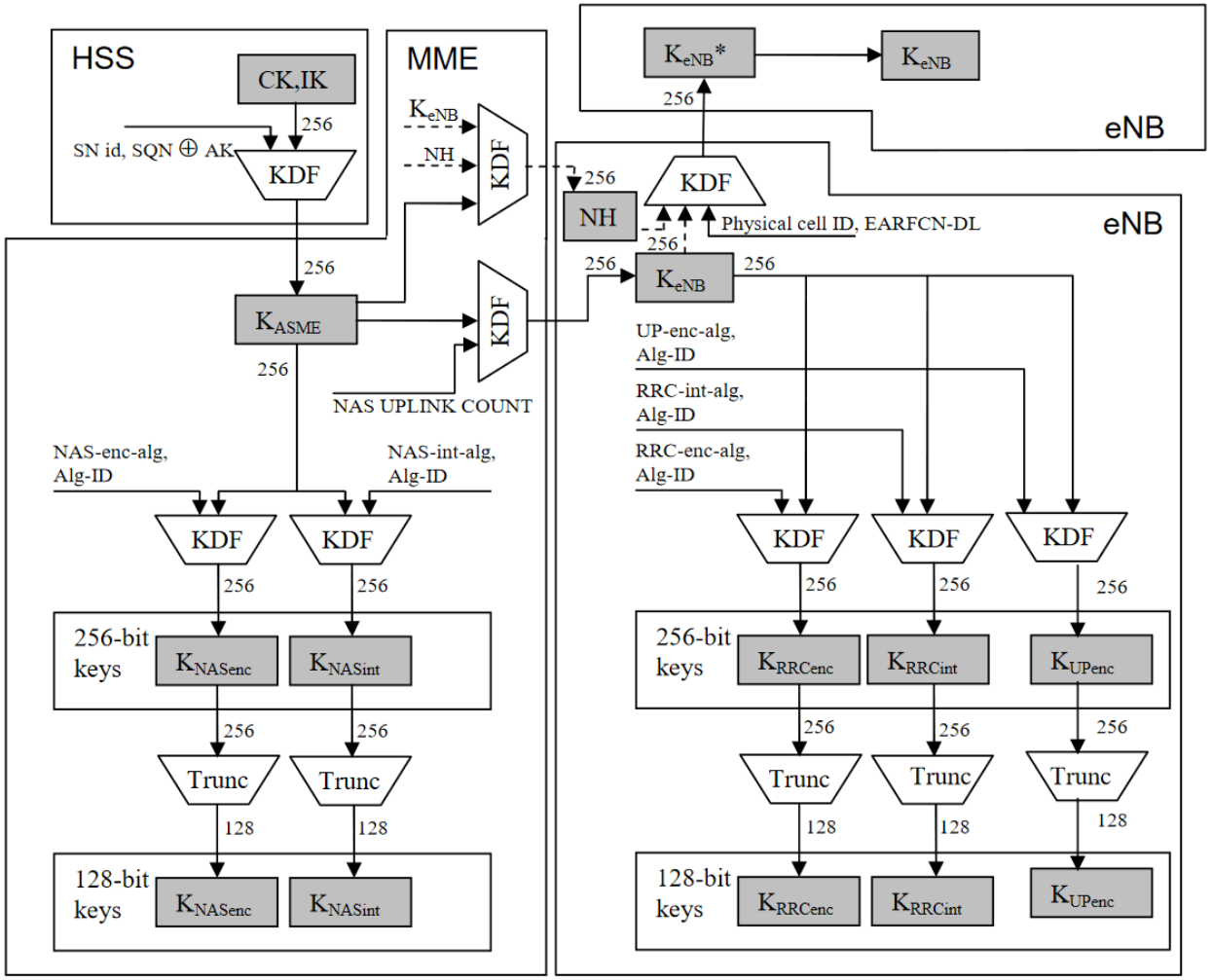
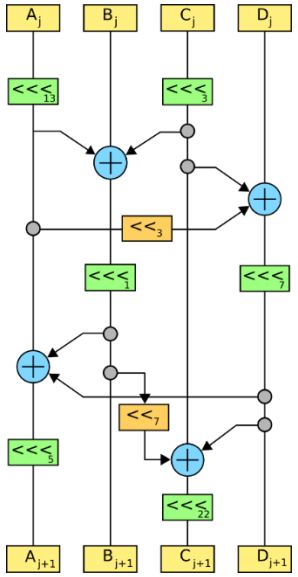


Figure 6.2-2: Key distribution and key derivation scheme for EPS (in particular E-UTRAN) for network nodes.



END OF PART 1
(SYMMETRIC CRYPTO)

Summary of symmetric cryptography

Primitive	Functionality + syntax	Security goal	Acronym	Examples
Pseudorandom function	Keyed function mapping fixed-length input to fixed-length output $F : \mathcal{K} \times \{0,1\}^{\text{in}} \to \{0,1\}^{\text{out}}$	Indistinguishability from random function	PRF	AES HMAC
Block cipher / pseudorandom permutation	Encrypt fixed-length block $E : \mathcal{K} \times \{0,1\}^{n} \to \{0,1\}^{n}$	Indistinguishability from random permutation	PRP	AES
Encryption	Encrypt variable-length input $\text{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ $\text{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{M} \to \mathcal{C}$ (nonce-based)	Confidentiality: attacker should learn nothing about plaintext (except length) from ciphertexts	IND-CPA IND-CCA	CTR CBC$
MAC	Produce fixed-length tag on variable-length message $\text{Tag} : \mathcal{K} \times \mathcal{M} \to \mathcal{T}$ $\text{Vrfy} : \mathcal{K} \times \mathcal{M} \times \mathcal{T} \to \{\text{Valid}, \text{Invalid}\}$	Integrity: attacker shouldn't be able to forge messages, i.e., create new messages with valid tags	UF-CMA	CBC-MAC CMAC HMAC
Authenticated encryption	Encrypt variable-length input $\text{Enc} : \mathcal{K} \times \mathcal{M} \to \mathcal{C}$ $\text{Dec} : \mathcal{K} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$ With associated data + nonces (AEAD) $\text{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{M} \to \mathcal{C}$ $\text{Enc} : \mathcal{K} \times \mathcal{N} \times \mathcal{A} \times \mathcal{C} \to \mathcal{M} \cup \{\perp\}$	Confidentiality + ciphertext integrity Confidentiality (message) + ciphertext and AD integrity	AE	EtM GCM OCB CCM
Hash function	Keyless function mapping variable-length input to fixed-length output $H : \mathcal{X} \to \mathcal{Y}$ $H : \{0,1\}^{*} \to \{0,1\}^{n}$	Collision-resistance + one-wayness		SHA1 SHA2-256 SHA2-512 SHA3

Summary of symmetric cryptography

Primitive	Functionality + syntax	Security goal	Acronym	Examples
Pseudorandom generator	Function mapping short input seed to long (basically infinite) output string $\mathrm{G} : \{0,1\}^{\ell} \rightarrow \{0,1\}^{L}$	Indistinguishability: output $G(s)$ should look like a random string in $\{0,1\}^{L}$	PRNG PRG	AES-CTR ChaCha20
Stream ciphers	Encryption schemes based on PRGs. Nonce or IV based.	IND-CPA		AES-CTR ChaCha20
Key derivation function	Expands high-entropy input to multiple uniform and independent keys	Entropy extraction + PRNG	KDF	HKDF
True random generator	Produce random bits	High min-entropy	TRNG	Ring Oscillators,