

CSE 207B: Applied Cryptography

Nadia Heninger

UCSD

Fall 2025 Lecture 2

Last time

- The one-time pad is perfectly secure.
- No cipher with $|K| < |M|$ can be perfectly secure.

Practical issues with the one-time pad

1. How to generate large amounts of perfectly uniform randomness?
2. How to distribute large amounts of key material, as much as the data you wish to send?

Relaxing perfect secrecy

Definition (Perfect secrecy)

(Enc, Dec) a cipher over (K, M, C) .

Let k be a random variable uniformly distributed over K .

$\forall m_0, m_1 \in M, \forall c \in C$:

$$\Pr[\text{Enc}_k(m_0) = c] = \Pr[\text{Enc}_k(m_1) = c]$$

Relaxing perfect secrecy

Definition (Perfect secrecy)

(Enc, Dec) a cipher over (K, M, C) .

Let k be a random variable uniformly distributed over K .

$\forall m_0, m_1 \in M, \forall c \in C$:

$$\Pr[\text{Enc}_k(m_0) = c] = \Pr[\text{Enc}_k(m_1) = c]$$

Ideas:

- Model a computationally bounded adversary
- Allow a small/negligible probability of failure
- Introduce a game-style security definition

More details

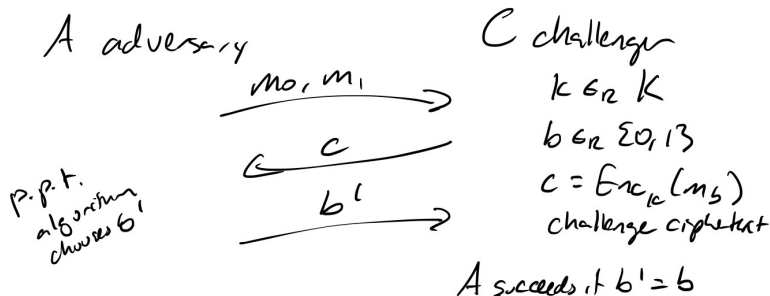
- Model a computationally bounded adversary
 - Theoretical viewpoint: probabilistic polynomial time algorithm
 - Practical viewpoint: Adversary has 2^{100} running time
- Allow a small/negligible probability of failure
 - Allow $|\Pr[f(\text{Enc}_k(m_0))] - \Pr[f(\text{Enc}_k(m_1))]| \leq \epsilon$
 - Theoretical viewpoint: $\epsilon(n)$ negligible if

$$\forall \text{ polynomial } p() \quad \exists N \text{ s.t. } \epsilon(n) \leq 1/p(n) \quad \forall n \geq N$$

(Examples: 2^{-n} , $2^{-\sqrt{n}}$, $n^{-\log n}$)

- Practical viewpoint: $\epsilon < 2^{-100}$.
- Introduce a game-style security definition
 - Game idealizes interaction between adversary and challenger

Semantic security



Definition (Semantic security)

A cipher (Enc, Dec) is semantically secure if for all efficient adversaries A , $|\Pr[A \text{ succeeds}] - \frac{1}{2}|$ is negligible.

Example

Let Enc be such that an adversary A can compute the least significant bit of m from $\text{Enc}(m)$.

Computationally relaxing the one-time pad

Recall one-time pad encryption:

$$\text{Enc}_k(m) = k \oplus m \quad \text{Dec}_k(c) = c \oplus k$$

Instead of requiring k to be truly random, replace k with some pseudorandom values generated by an algorithm G from a much shorter seed s :

$$\text{Enc}_s(m) = G(s) \oplus m \quad \text{Dec}_s(c) = c \oplus G(s)$$

Pseudorandom Generator

Definition (Pseudorandom generator (PRG))

G deterministic, polynomial time algorithm.

$$G : \{0, 1\}^{\ell} \rightarrow \{0, 1\}^L$$

- Expansion: $L > \ell$
- Pseudorandomness: No efficient algorithm D can distinguish $G(s)$ from a truly random string r with better than negligible advantage. ($r \in_R \{0, 1\}^L, s \in_R \{0, 1\}^{\ell}$)

$$|\Pr[D(r) = 1] - \Pr[D(G(s)) = 1]| < \epsilon$$

Motivation: statistical tests

How could you tell if a string was random or not?

Idea: Run some algorithm on the string and check if expected properties differ much from expectation.

- Count number of 1s
- Look at numbers of long runs
- Analyze sequences of consecutive digits
- Play 200000 games of craps and look at winning distribution

These are all examples from the “Diehard” tests.

Stream Cipher Encryption

- $\text{Enc}_k(m) = G(k) \oplus m = c$
- $\text{Dec}_k(c) = c \oplus G(k)$

Theorem

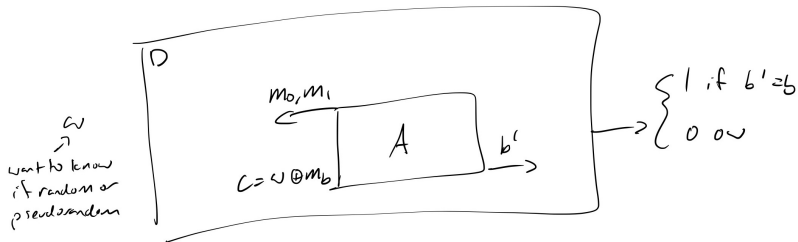
If G is a PRG then stream cipher encryption constructed from G is semantically secure.

Theorem

If G is a PRG then stream cipher encryption constructed from G is semantically secure.

Proof.

(By reduction.) Show if A can distinguish encryptions, then G is not a PRG.



- $\omega = \text{random}$: $\Pr[D(r) = 1] = 1/2$
- $\omega = G(s)$: $\Pr[D(G(s)) = 1] = \Pr[A \text{ succeeds}] = 1/2 + \delta$

Then $|\Pr[D(r) = 1] - \Pr[D(G(s)) = 1]| = \delta > \text{negl.}$ by assumption.

Security issues with one-time pad: "two-time" pad

- Theoretical viewpoint: Try to define security definition for multiple encryptions.

$$A \xrightarrow{(m_0, m_1), (m_0', m_1')} C$$
$$c = \text{Enc}_k(m_0^b), \text{Enc}_k(m_1^b)$$

←

Security issues with one-time pad: "two-time" pad

- Theoretical viewpoint: Try to define security definition for multiple encryptions.

$$A \xrightarrow{(m_0, m_1), (m_0', m_1')} C$$
$$c = \text{Enc}_k(m_0^b), \text{Enc}_k(m_1^b)$$

←

There are semantically secure encryption schemes that are not secure under this definition.

Security issues with one-time pad: "two-time" pad

- Theoretical viewpoint: Try to define security definition for multiple encryptions.

$$A \xrightarrow{(m_0, m_1), (m_0', m_1')} C$$
$$c = \text{Enc}_k(m_0^b), \text{Enc}_k(m_1^b)$$

←

There are semantically secure encryption schemes that are not secure under this definition.

$(\text{Enc}_k(0^n), \text{Enc}_k(0^n))$ vs. $(\text{Enc}_k(0^n), \text{Enc}_k(1^n))$

Security issues with one-time pad: "two-time" pad

- Theoretical viewpoint: Try to define security definition for multiple encryptions.

$$A \xrightarrow{(m_0, m_1), (m_0', m_1')} C$$
$$c = \text{Enc}_k(m_0^b), \text{Enc}_k(m_1^b)$$

←

There are semantically secure encryption schemes that are not secure under this definition.

$(\text{Enc}_k(0^n), \text{Enc}_k(0^n))$ vs. $(\text{Enc}_k(0^n), \text{Enc}_k(1^n))$

- Practical viewpoint: $c_0 = m_0 \oplus G(s)$, $c_1 = m_1 \oplus G(s)$

$$c_0 \oplus c_1 = m_0 \oplus m_1$$

Security issues with one-time pad: malleability

Let $c = \text{Enc}_k(m) = m \oplus G(k)$.

Then $\text{Dec}_k(c \oplus d) = (m \oplus G(k)) \oplus d \oplus G(k) = m \oplus d$.

Stream ciphers in practice: RC4

1987 Rivest Cipher 4; trade secret of RSA corporation

1994 Code posted anonymously to cipherpunks mailing list, sci.crypt newsgroup. Used in SSL/TLS, WEP, WPA, Kerberos, etc.

2001 Fluhrer, Mantin, Shamir

- Weaknesses in key scheduling algorithm
- Repeated initialization vectors allow break
- 2nd byte of keystream is biased
- Used to crack WEP

2013 AlFardan et al. attack on RC4 in TLS to steal cookies with 2^{34} ciphertexts.

1. Empirically measure probabilities for each byte.
2. Encrypt fixed unknown plaintext with many keys, compare to distribution.
3. Return message maximizing probabilities.



On the Security of RC4 in TLS

Nadhem AlFardan, Dan Bernstein, Kenny Paterson, Bertram Poettering, Jacob Schuldt

Royal Holloway, University of London
University of Illinois at Chicago

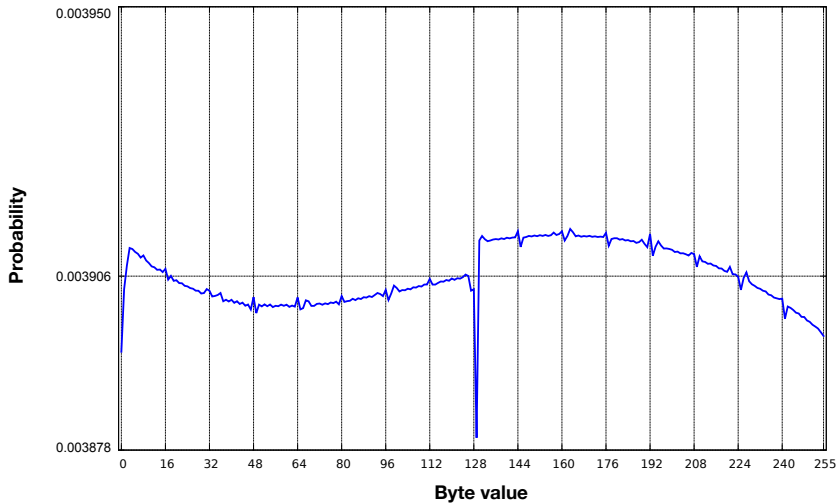
<http://www.isg.rhul.ac.uk/tls/>



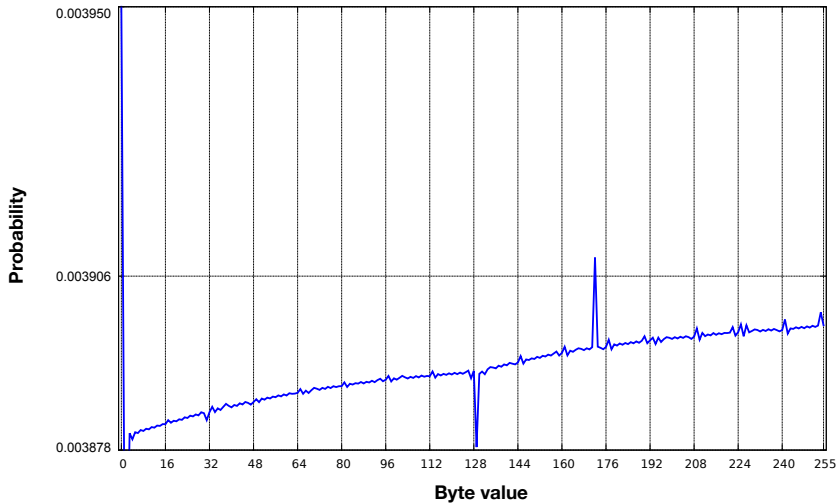
Information Security Group

UIC

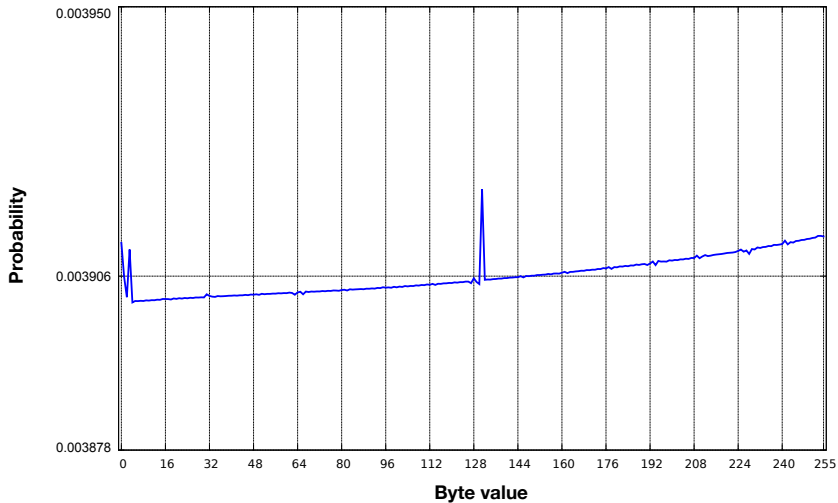
Keystream Distribution at Position 1



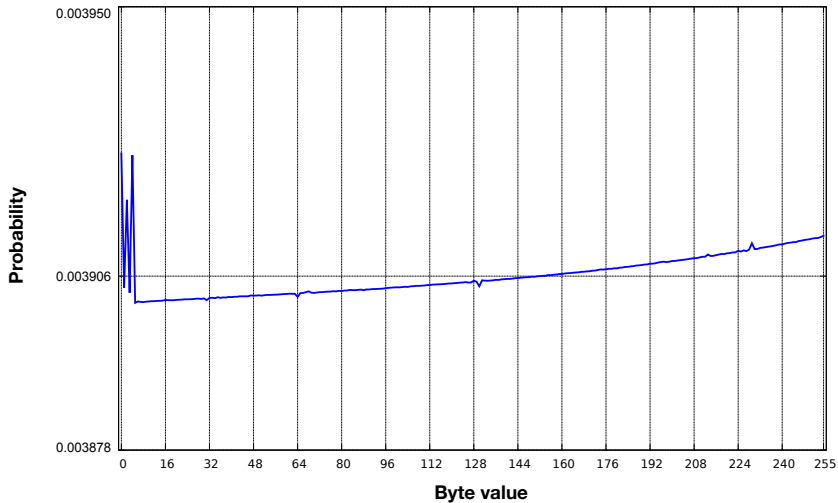
Keystream Distribution at Position 2



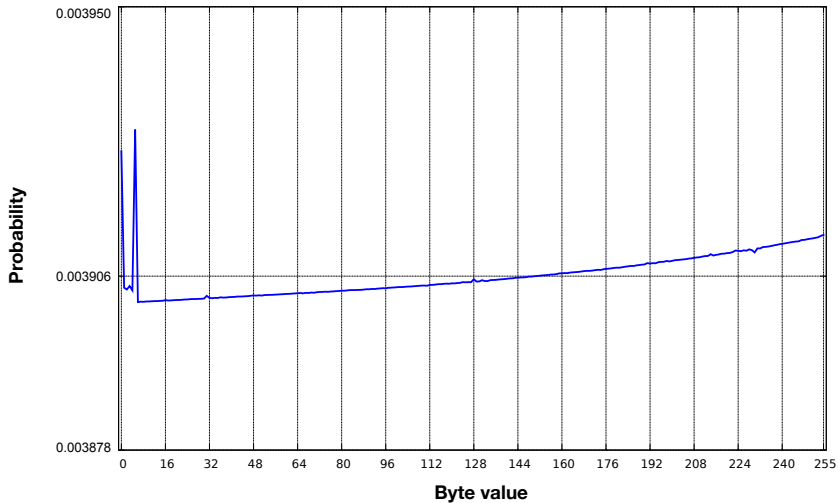
Keystream Distribution at Position 3



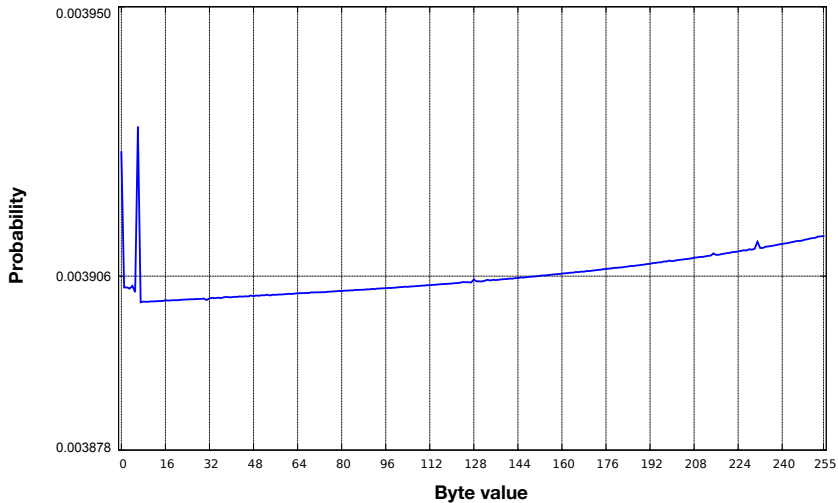
Keystream Distribution at Position 4



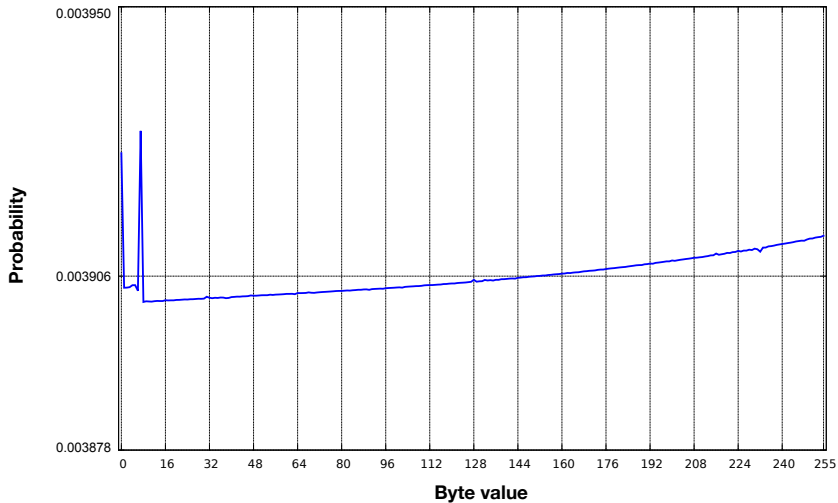
Keystream Distribution at Position 5



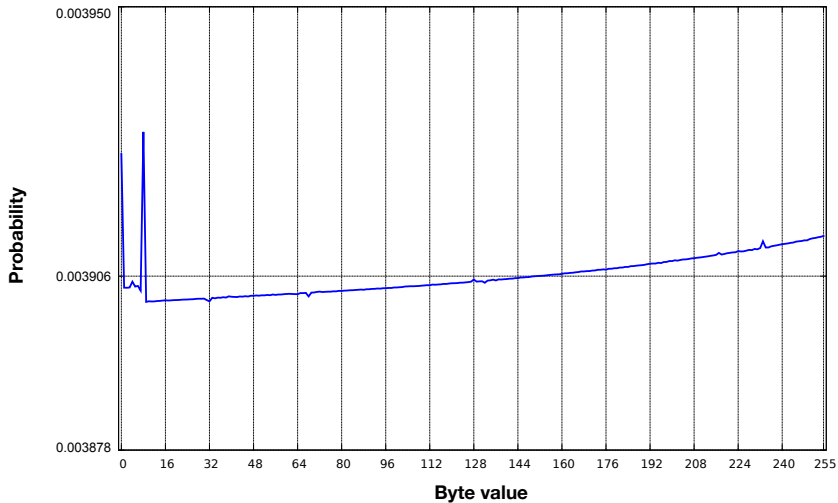
Keystream Distribution at Position 6



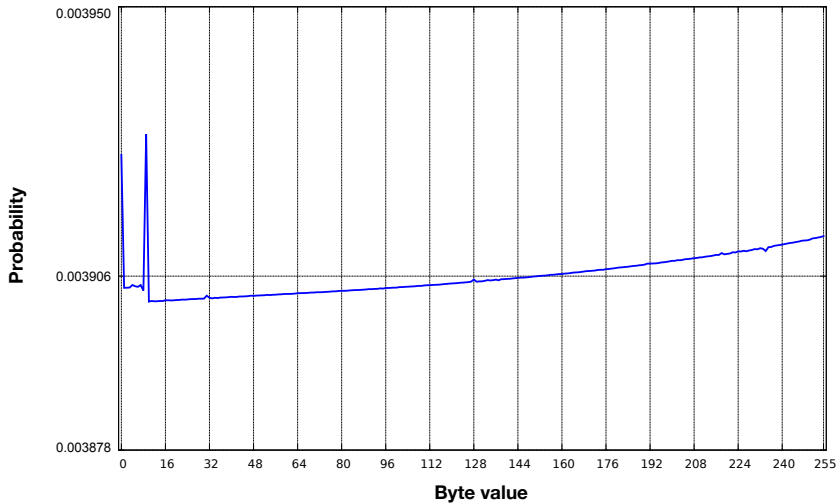
Keystream Distribution at Position 7



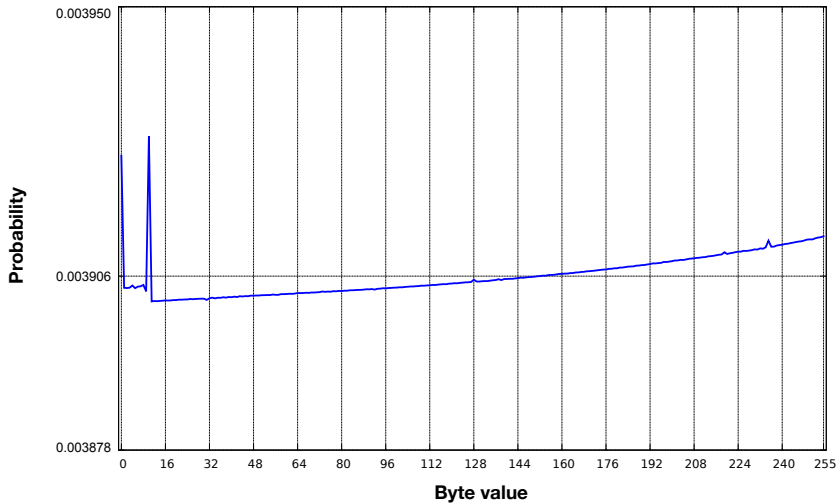
Keystream Distribution at Position 8



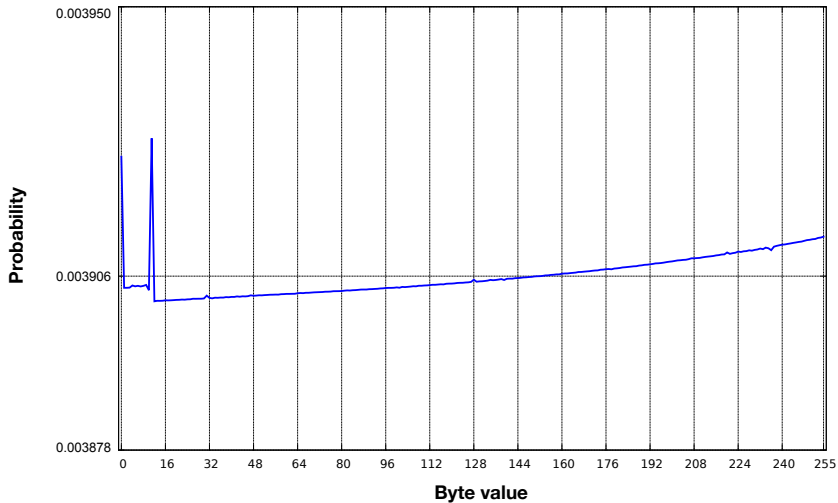
Keystream Distribution at Position 9



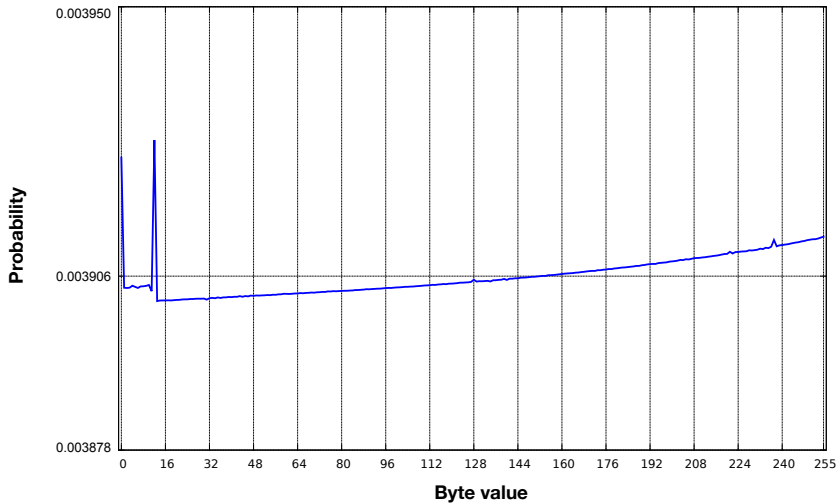
Keystream Distribution at Position 10



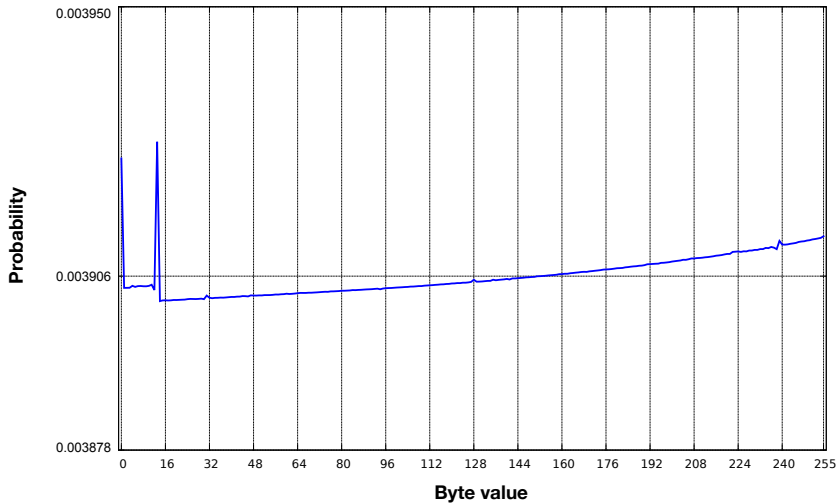
Keystream Distribution at Position 11



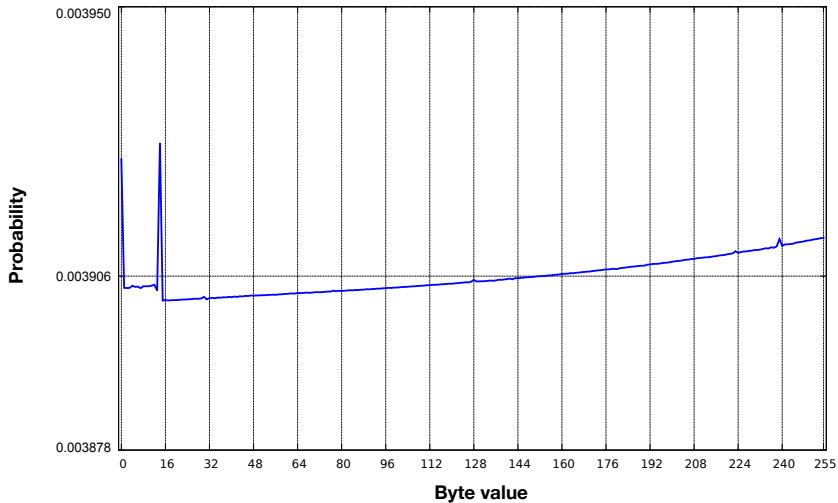
Keystream Distribution at Position 12



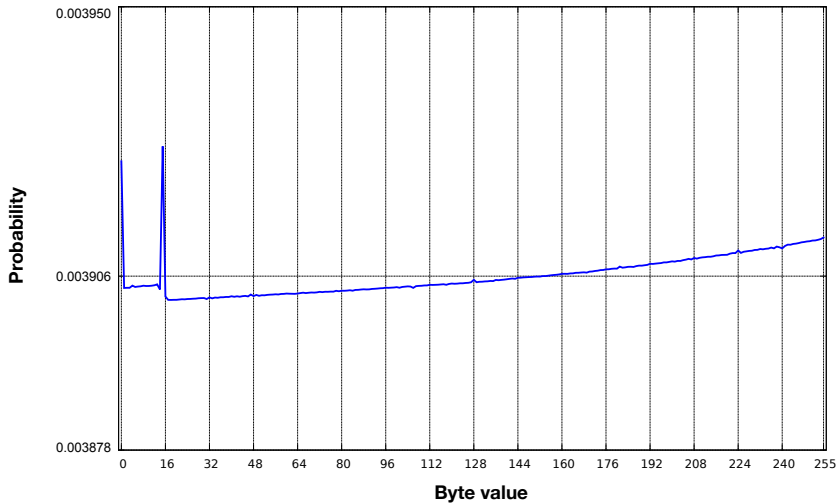
Keystream Distribution at Position 13



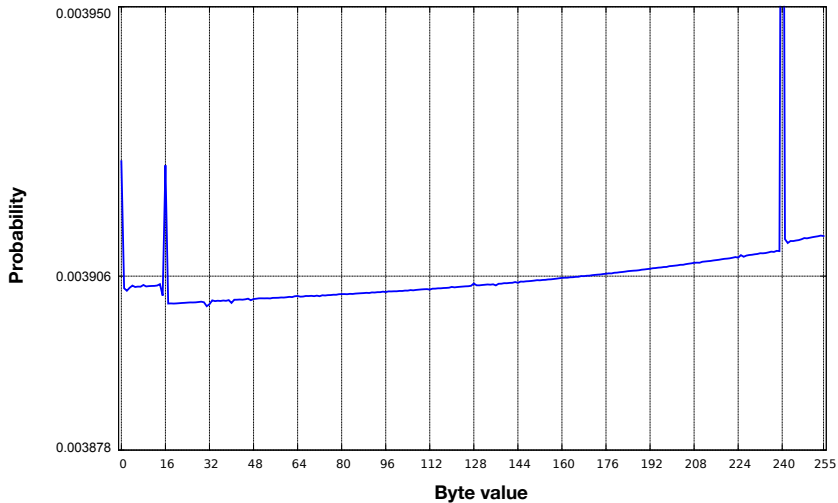
Keystream Distribution at Position 14



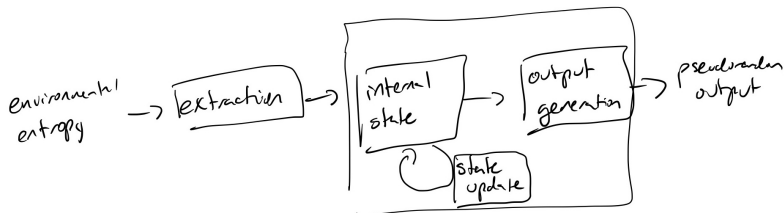
Keystream Distribution at Position 15



Keystream Distribution at Position 16



A PRG isn't quite a *pseudorandom number generator*



There are multiple properties we want out of a *pseudorandom number generator* that aren't captured by our PRG definition:

- Being able to continuously add entropy to RNG internal state
- Handling nonuniform inputs from the environment without becoming insecure

Linux's `/dev/random` is a software random number generator that takes entropy from keyboard, mouse, hardware interrupt timings.

Intel's `RdRand` instruction implements a PRNG in hardware that takes input from a hardware random source.