18733: Applied Cryptography Recitation

Asymptotic Security and Cryptographic Design

Gihyuk Ko

February 3, 2017

Carnegie Mellon University

Asymptotic Notations

Introduced in algorithmic complexity theory

- Used to simplify notion of complexity of solving a problem
- Categorizing different problems into similar groups

Notations

- $O(n)$: $f(n) = O(g(n))$ if $f(n)/g(n)$ is bounded as $n \to \infty$
- $o(n)$: $f(n) = o(g(n))$ if $f(n)/g(n) \to 0$ as $n \to \infty$
- $\Omega(n)$: $f(n) = \Omega(g(n))$ if f is an upper bound for g
- $\Theta(n)$: $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Theta(g(n))$

ex) *polynomial time algorithm*: an algorithm that solves in $O(n^k)$ time for some integer k

Asymptotic Formalization of Security

“Provable security is Asymptotic”

Security parameter 1^n: n is chosen beforehand, may be the message length

a parameter we choose to prove security on

Adversary: *polynomial time algorithm* in n

Negiligible function $\epsilon : \mathbb{N} \rightarrow [0,1]$

- $\epsilon(n)$ is negligible if $\forall c, \exists n_0$ s.t. $\epsilon(n) < 1/n^c$ for all $n > n_0$
- $\epsilon(n) = O(1/n^c)$ for all c (same with $\epsilon(n) = o(1/n^c)$ for all c)

Proof of security: adversary’s advantage is *negiligible*!

Cryptographic Design: Motivation

A good thing: we have definitions for security

- Encryption: Semantic security (e.g. `IND-CPA`, `IND$-CPA`)
- PRNG: Statistical tests

How do we actually build functions that satisfy these security properties?
Given a function, how can we test that it satisfies these properties?

Cryptographic Design: Goals

Build a function that …

… is secure …

- No *PPT* adversary can have *non-negligible advantage*

… and usable.

- Easily computable
- Short keys
- Compact software representation
- Compact hardware representation
- Parallelly computable
- Efficient on a wide range of platforms it might be deployed on

Case Study: Salsa20

Salsa20: Created by Daniel Bernstein in 2005

← This guy!

What is Salsa20?

Encryption Function (aka Snuffle 2005): $\{0,1\}^{256}\{0,1\}^{2^{70}} \to \{0,1\}^{2^{70}}$

Inputs

- 256-bit key k (secret)
- Message m such that $|m| \leq 2^{70}$

Outputs

- Ciphertext c such that $|c| = |m|$

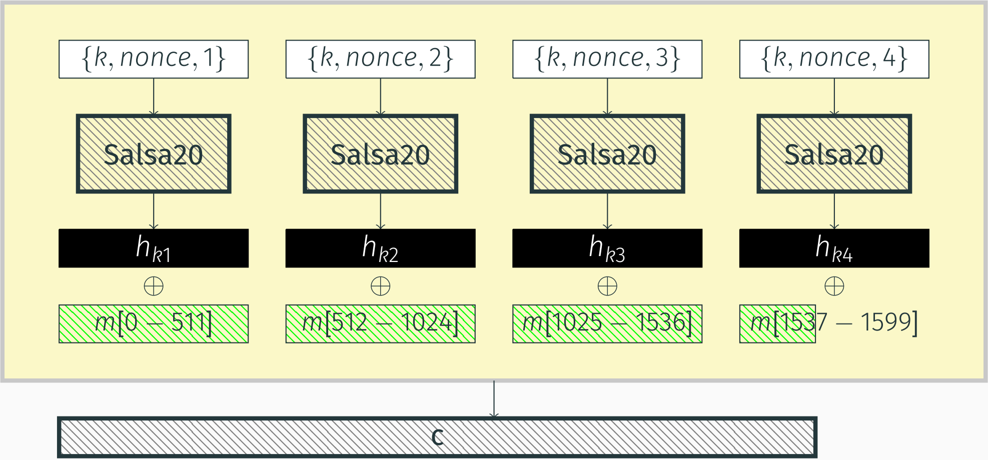
Salsa20 Core: $\{0,1\}^{512} \to \{0,1\}^{512}$

Input: 512-bit

- 256-bit key k (secret)
- 64-bit *nonce* (public), 64-bit *counter*, 128-bit fixed word

Output: 512-bit

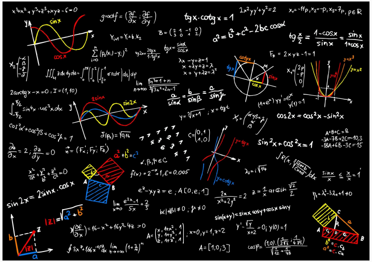
Salsa20: Encryption



Cryptographic Design: Salsa20

What should the “Salsa20 Core” boxes look like?

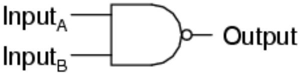
- Should it be very complex/complicated?
- Should they be extremely simple?



Functional Completeness

Every computable function (by a Turing Machine, i.e. not by Quantum Computer) can be expressed as a series of NAND-gates

NAND gate



A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Should there be multiplication?

Multiplication Seems like a simple operation

Many processors do not have very quick multiplication implementations

Some processors have *timing leaks* with multiplication

- Motorola PowerPC 8450 (G4e): 2 cycles normally, 1 cycle if 15 msb of operand are all 0s or 1s

Should there be S-Boxes?

S-box: Arbitrary mapping between some inputs and outputs via pre-defined lookup table

Due to memory restrictions, they can only support 8 bit operations

- Several lookups required to mangle 32 bits

Can introduce *timing attacks* due to cache interactions

Timing Attack

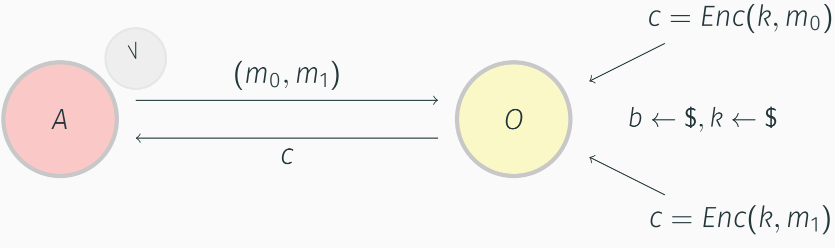
Crypto gets deployed in many settings

- Software library on personal computer
- Hardware on commercial processors
- Specialized payment processing equipment – Aircraft equipment
- Military applications
- … and thousands of other environments

Some obscure unforseen implementations might leak private information

Timing Attack

Calling a real world cryptographic function introduces side channels



$$Adv(A) = \Pr[A(Enc(k, m_0)) = 1] - \Pr[A(Enc(k, m_1)) = 1]$$

Timing Attack: Example

Let $f : \{0,1\}^n \to \{0,1\}^n$ be an encryption function where the runtime is equal to:

- 1ms * (number of positions where key and message are both 1)

Attack: input messages starting with $0 \cdots 01$, checking runtimes to figure out bits of the key are 1 and what bits are 0

Other Side Channels

- Power consumption
- Heat
- Noise
- Memory latency
- Cache timings
- … and many others

Side channels can vary a lot and very domain-specific

Add-Rotate-XOR (ARX) Operations

Add $\boxplus$: n-bit addition mod 2^n

$$10010101_{(2)} \boxplus 11110110_{(2)} = 10001011_{(2)}$$

Rotate $<<<$: constant-distance rotation operations

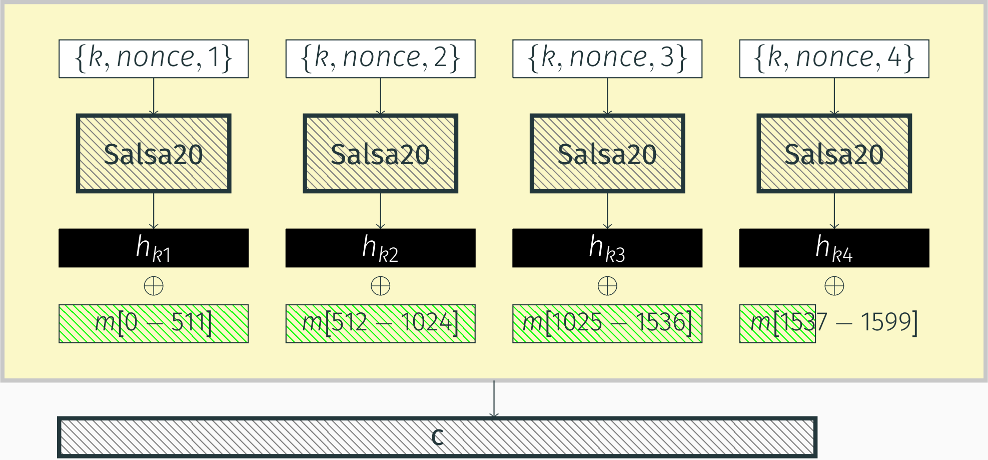
$$10010101_{(2)} <<< 1 = 00101011_{(2)}$$

XOR $\oplus$: bitwise addition

$$10010101_{(2)} \oplus 11110110_{(2)} = 01100011_{(2)}$$

These operations are immune to timing attacks!!

Salsa20: Encryption



Salsa20 Core

0x61707865	k[0,31]	k[32,63]	k[64,95]
k[96,127]	0x3320646e	nonce[0,31]	nonce[32,63]
ctr[0,31]	ctr[32,63]	0x79622d32	k[128,159]
k[160,191]	k[192,223]	k[224,255]	0x6b206574

Round: Let $b_n = n$ below diagonal

For each column do:

1. $b_1 \oplus = ((b_3 \boxplus b_4) <<< 7)$
2. $b_2 \oplus = ((b_4 \boxplus b_1) <<< 9)$
3. $b_3 \oplus = ((b_1 \boxplus b_2) <<< 13)$
4. $b_4 \oplus = ((b_2 \boxplus b_3) <<< 18)$

Transpose the matrix

Design Choices

Why the particular rotation distances?

> *"I chose the Salsa20 rotation distances 7, 9, 13, 18 as doing a good job of spreading every low-weight change across bit positions within a few rounds. The exact choice of distances doesn't seem very important."*

Why not interchange the addition and XOR?

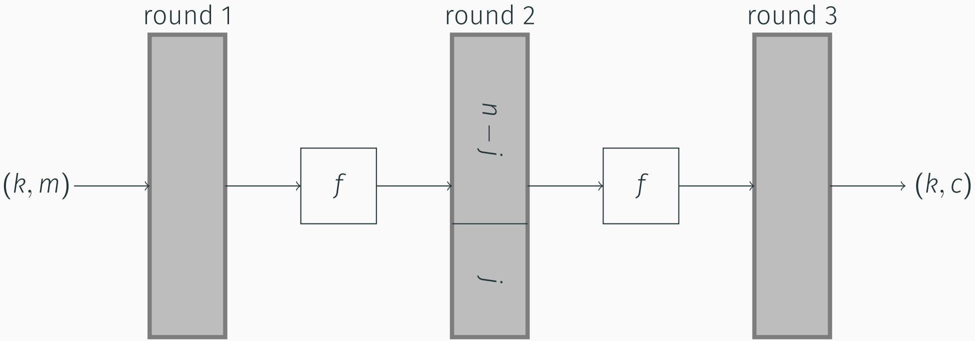
> *"I chose "xor a rotated sum" over "add a rotated xor" for simple performance reasons: the x86 architecture has a three-operand addition (LEA) but not a three-operand xor."*

Is Salsa20 Secure?

- Can we tell that our choice of function is really secure?
- Cryptanalysis is required to see if the function is secure against several 'well-known-attacks'

via XKCD

Meet in the Middle Attack



If we can find j bits in the middle of the cipher such that they do not depend on the l bits of the k-bit key, then we can reduce complexity of the exhaustive search attack to $2^{k-l} + 2^l$.

Salsa20 Attacks

- Currently best known attack breaks 8 rounds
- Differential variant of meet-in-the-middle attack: truncated differential cryptanalysis
  - 2165-operation attack on Salsa20/5 by Crowley
  - Aumasson, Fischer, Khazaei, Meier, and Rechberger reported a 2249-operation attack on Salsa20/8 and a 2153-operation attack on Salsa20/7.

Takeaways

Design of cryptographic implementations is very heuristic

- Needs to work well on current hardware
- Needs to be very fast
- Should be designed to resist known attacks
- Security is never really proved, just argued for, primitive is subject to attacks
- Proofs exist but rely on unproven (but thought to be safe) assumptions

Questions?