ChaCha20 Implementation

Focus on Security

Le Duy Nguyen

Faculty of Information Technology
Brno University of Technology

20/11/2023

ChaCha20

- Developed by Daniel J. Bernstein in 2008
- **Stream cipher** with 256-bit key and 96-bit nonce
- Recommended by the Czech National Cyber and Information Security Agency (NÚKIB)

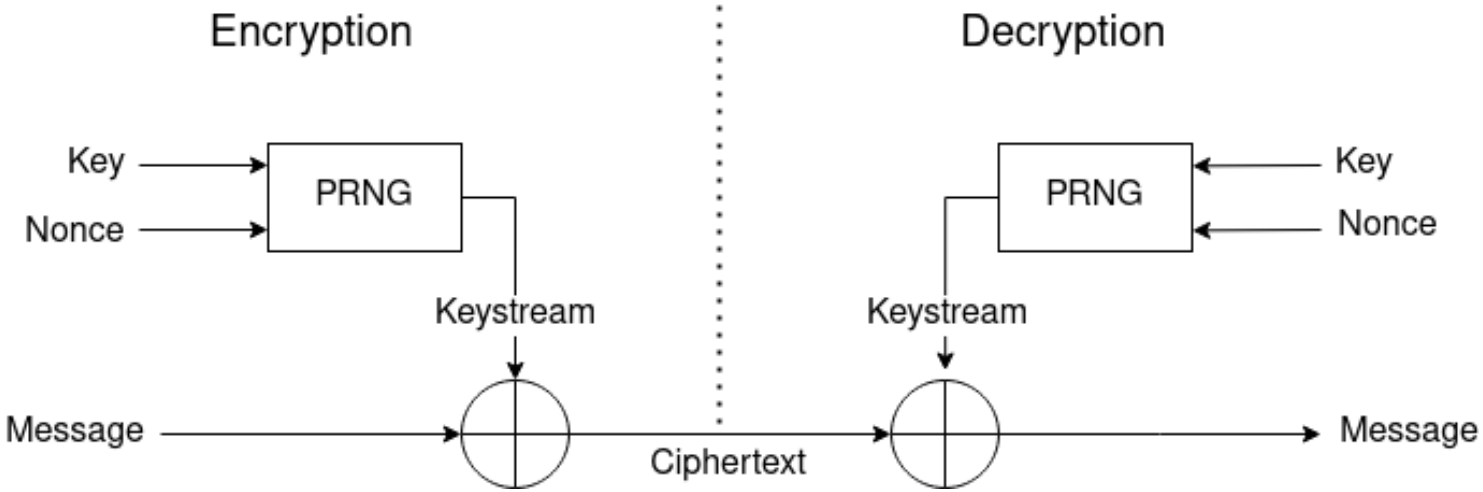
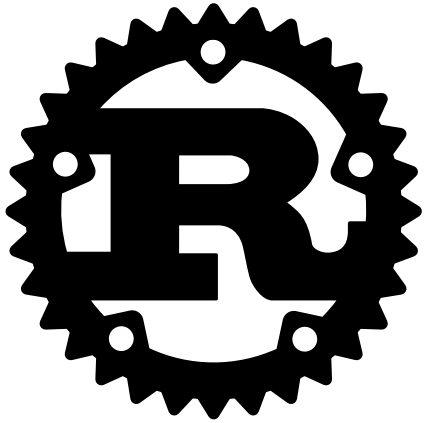


Figure 1: ChaCha20's Scheme

Technology

- Programming language: **Rust**
- Architecture: **x86** family



- **Github**: https://github.com/tmokenc/VUT-FIT-SCO

Secure Implementation Guideline

- Sensitive functions **must** have time complexity of $O(1)$
  - Operations like `Divide`, `Modulo`, `If` introduce variable time complexity.

```
x = if s { a } else { b }
```

```
x = b & ((-(s & 1)) & (a ^ b))
```

-

-

-

Secure Implementation Guideline

- Sensitive functions **must** have time complexity of $O(1)$
  - Operations like `Divide`, `Modulo`, `If` introduce variable time complexity.

```
x = if s { a } else { b }
```

```
x = b & ((-(s & 1)) & (a ^ b))
```

- **Clean up** data when no longer needed
-
-

Secure Implementation Guideline

- Sensitive functions **must** have time complexity of $O(1)$
  - Operations like `Divide`, `Modulo`, `If` introduce variable time complexity.

```
x = if s { a } else { b }
```

```
x = b & ((-(s & 1)) & (a ^ b))
```

- **Clean up** data when no longer needed
- Minimal dependencies
-

Secure Implementation Guideline

- Sensitive functions **must** have time complexity of $O(1)$
  - Operations like `Divide`, `Modulo`, `If` introduce variable time complexity.

```
x = if s { a } else { b }
```

```
x = b & ((-(s & 1)) & (a ^ b))
```

- **Clean up** data when no longer needed
- Minimal dependencies
- Testing!

Problem with Rust

Rust is **too smart** for us crypto-developers

What we want

```
let mut key = [1, 2, 3];

do_something_with_key(&key);
key = [0, 0, 0]; // clean up
```

What Rust actually does

```
let mut key = [1, 2, 3];

do_something_with_key(&key);
// key = [0, 0, 0]; clean up
```

⬆ *"Bro! You have **deadcode** here. I removed it for you. No need to thank me."*

Problem with Rust 2

Rust is **too smart** for us crypto-developers

Remember this trick?

```
x = if s { a } else { b }
```

```
x = b & ((-(s & 1)) & (a ^ b))
```

They are the same in x86 assembly

```
test   dil,dil
jne    8   ; conditional jump
mov    sil,dl
mov    eax,esi
ret
```

↳ Solution: `core::hint::black_box`

ChaCha20: Usage

Input

- Message **or** Ciphertext
- 256-bit Key + 96-bit Nonce

Output

- Ciphertext **or** Message

```
use chacha20poly1305::ChaCha20;
let message = b"Your message or cipher text";
let (key, nonce) = cryptographic_rng();
let mut cipher = ChaCha20::new(key, nonce);
match cipher.perform(message) {
  Ok(ciphertext) => // Success
  Err(_) => // The data is too long
}
```

ChaCha20-Poly1305

AEAD Construction

AEAD Construction

- ***A***uthenticated ***E***ncryption with ***A***ssociated ***D***ata
  - ChaCha20 provides **confidentiality** and **authentication**
  - what about **integrity**?
- **Poly1305** - MAC function
  - developed by the same author, Daniel J. Bernstein
  - works **seamlessly** with ChaCha20
- Recommended by the Czech National Cyber and Information Security Agency (NÚKIB)
- **ChaCha20-Poly1305** is used in various applications, including OpenSSH and TLS 1.3

AEAD Construction: Scheme

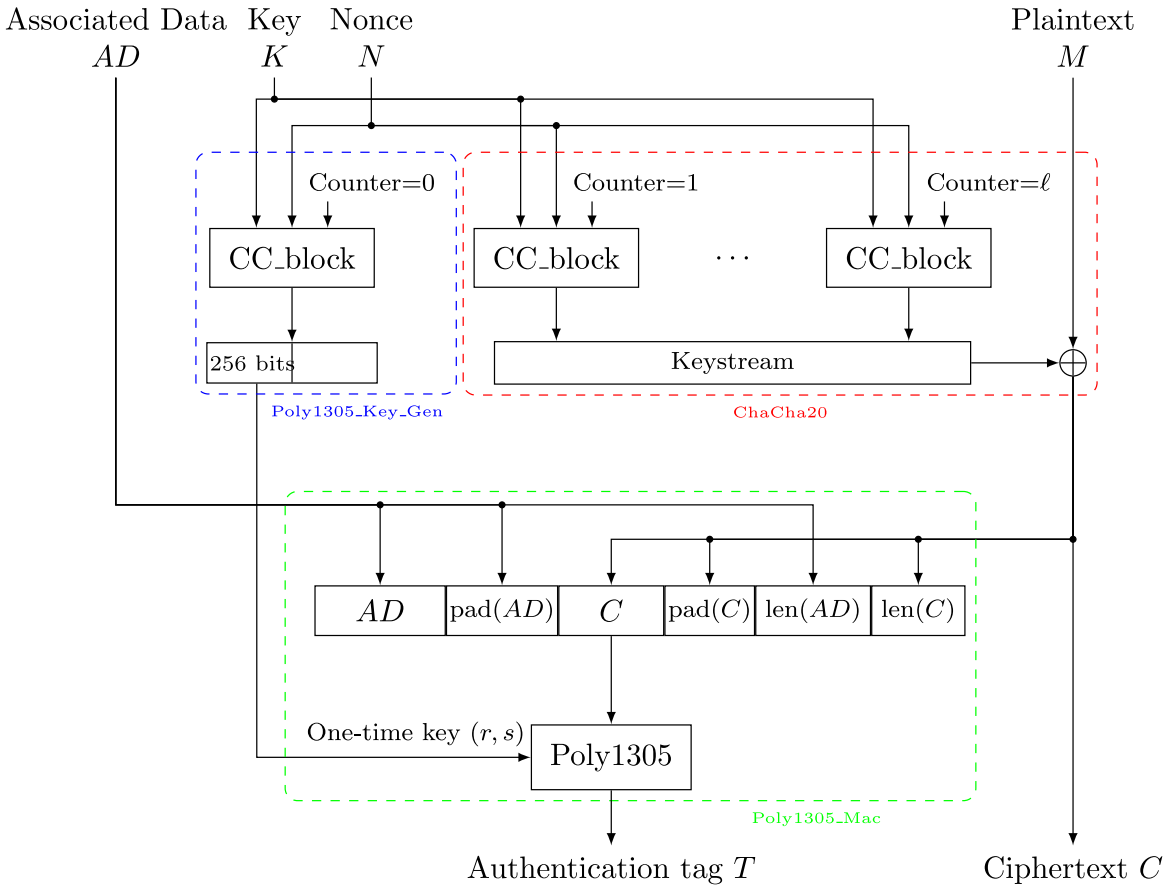


Figure 3: ChaCha20-Poly1305 AEAD construct.

Usage: Encryption

Input

- Message
- 256-bit Key + 96-bit Nonce
- AAD

Output

- Ciphertext
- Authentication Tag

```
use chacha20poly1305::ChaCha20Poly1305;
let message = b"Your message";
let (key, nonce, aad) = cryptographic_rng();
let cipher = ChaCha20Poly1305::new(key, nonce, aad);
match cipher.encrypt_oneshot(message) {
    Ok((ciphertext, tag)) => // Succesfully
    Err(_) => // Something went wrong
#}
```

Usage: Decryption

Input

- Message + Authentication Tag
- 256-bit Key + 96-bit Nonce
- AAD

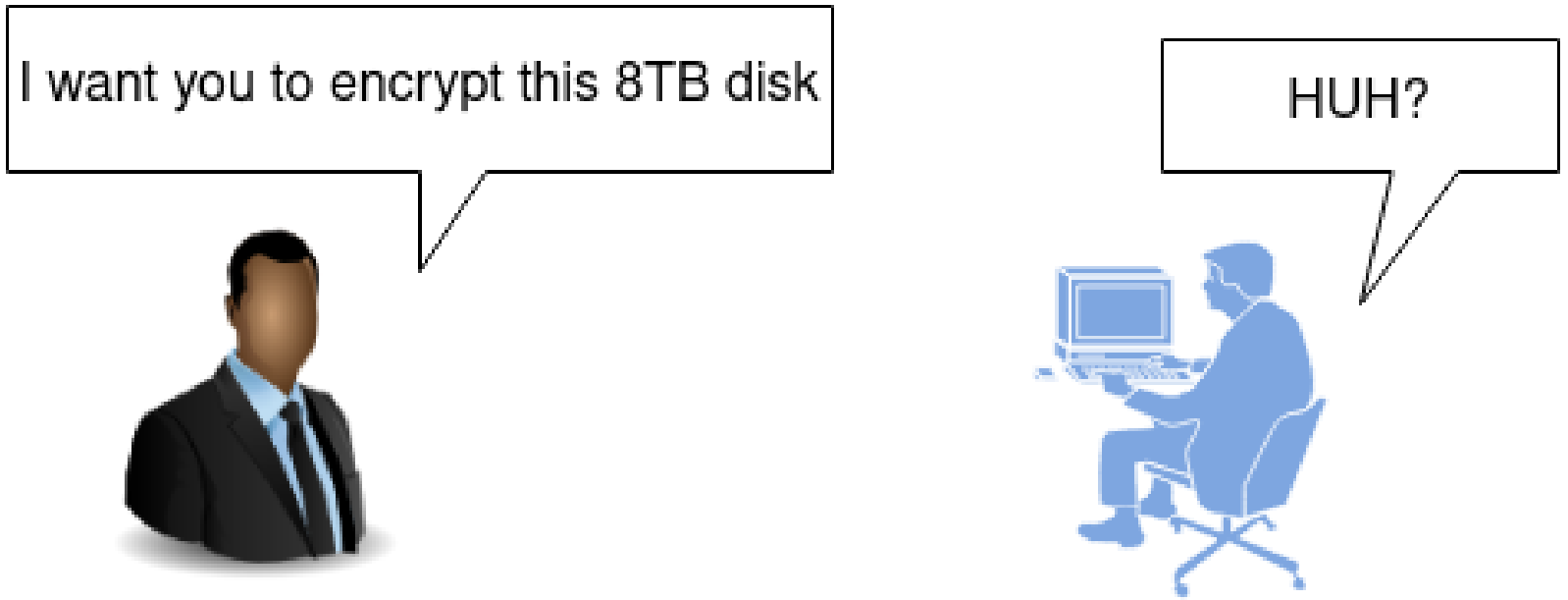
Output

- Authenticated Message

```
use chacha20poly1305::ChaCha20Poly1305;
let (key, nonce, aad, ciphertext, tag) = receive();
let cipher = ChaCha20Poly1305::new(aad, key, nonce);
match cipher.decrypt_oneshot(ciphertext, tag) {
  Ok(authenticated_message) => // Success
  Err(why) => // Something wrong with the message
}
```

Limitation

- Limited to encrypting messages up to **256 GiB** in size.



Security Analysis

- Timing Attack
  The execution time is $O(n)$ where n is the length of message
  ↳ have better luck with brute-force
- Power Analysis
  ↳ *Unfortunately*, the current implementation is vulnerable to power analysis

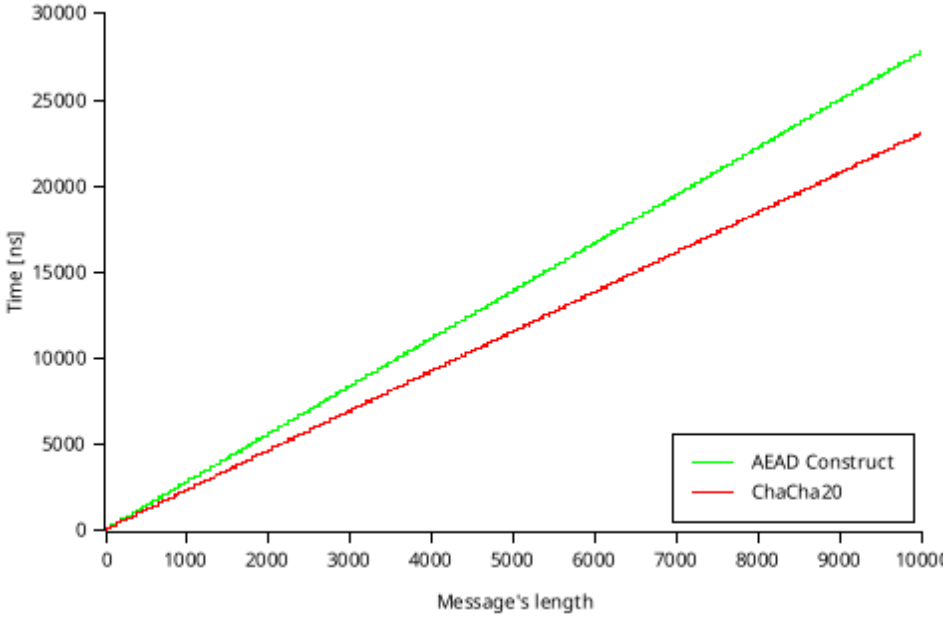


Figure 5: Execution time / message's length

Thank you for your attention.

Sources

- https://github.com/tmokenc/VUT-FIT-SCO

- https://en.wikipedia.org/wiki/ChaCha20-Poly1305
- https://datatracker.ietf.org/doc/html/rfc8439