Engineering Cryptographic Software

A brief introduction to ChaCha20

Peter Schwabe

Winter 2026/27

Symmetric Crypto Overview (highly simplified)



- Block ciphers: DES, AES
- Hash functions: SHA2, SHA3/SHAKE
- Authenticators: HMAC, GHASH, Poly1305
- Stream ciphers. . .

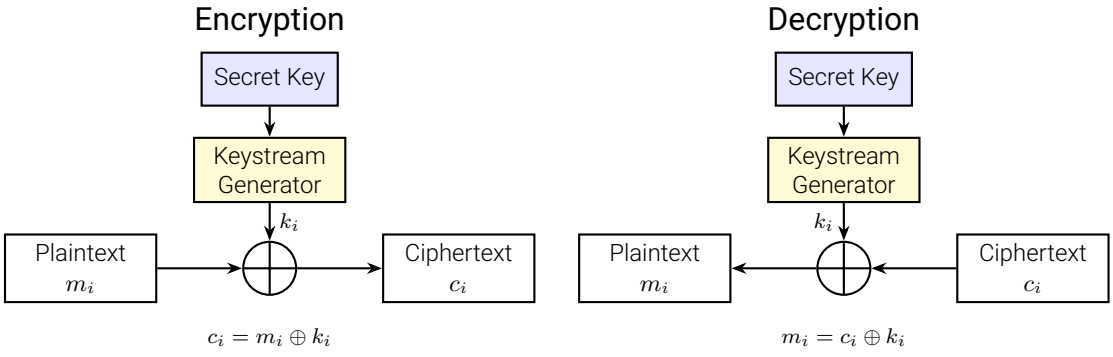
Stream Cipher Recap: General



- Symmetric key cipher
- General idea: Combine plaintext with a stream of pseudorandom bytes, the *keystream*
- Loosely inspired by the one-time pad (OTP)

Stream Cipher Recap: General

- Symmetric key cipher
- General idea: Combine plaintext with a stream of pseudorandom bytes, the *keystream*
- Loosely inspired by the one-time pad (OTP)



From RC4 to eSTREAM



- Historically: two widely deployed stream ciphers:
  - AES in streaming mode (AES-CTR)
  - RC4

From RC4 to eSTREAM



- Historically: two widely deployed stream ciphers:
  - AES in streaming mode (AES-CTR)
  - RC4
- Block cipher AES problematic in software implementations
  - Table-based approach has “textbook” cache-timing vulnerabilities
  - Alternative approaches (bitslicing) not fast on all platforms

From RC4 to eSTREAM



- Historically: two widely deployed stream ciphers:
  - AES in streaming mode (AES-CTR)
  - RC4
- Block cipher AES problematic in software implementations
  - Table-based approach has “textbook” cache-timing vulnerabilities
  - Alternative approaches (bitslicing) not fast on all platforms
- RC4 broken by Fluhrer, Mantin, and Shamir in 2001

From RC4 to eSTREAM



- Historically: two widely deployed stream ciphers:
  - AES in streaming mode (AES-CTR)
  - RC4
- Block cipher AES problematic in software implementations
  - Table-based approach has “textbook” cache-timing vulnerabilities
  - Alternative approaches (bitslicing) not fast on all platforms
- RC4 broken by Fluhrer, Mantin, and Shamir in 2001
- More broken by Klein in 2005

From RC4 to eSTREAM



- Historically: two widely deployed stream ciphers:
  - AES in streaming mode (AES-CTR)
  - RC4
- Block cipher AES problematic in software implementations
  - Table-based approach has “textbook” cache-timing vulnerabilities
  - Alternative approaches (bitslicing) not fast on all platforms
- RC4 broken by Fluhrer, Mantin, and Shamir in 2001
- More broken by Klein in 2005
- Broken even more by AlFardan, Bernstein, Paterson, Poettering, Schuldt in 2013

From RC4 to eSTREAM



- Historically: two widely deployed stream ciphers:
  - AES in streaming mode (AES-CTR)
  - RC4
- Block cipher AES problematic in software implementations
  - Table-based approach has “textbook” cache-timing vulnerabilities
  - Alternative approaches (bitslicing) not fast on all platforms
- RC4 broken by Fluhrer, Mantin, and Shamir in 2001
- More broken by Klein in 2005
- Broken even more by AlFardan, Bernstein, Paterson, Poettering, Schuldt in 2013
- . . . and even more by Garman, Paterson, and van der Merwe in 2015

From RC4 to eSTREAM



- Historically: two widely deployed stream ciphers:
  - AES in streaming mode (AES-CTR)
  - RC4
- Block cipher AES problematic in software implementations
  - Table-based approach has “textbook” cache-timing vulnerabilities
  - Alternative approaches (bitslicing) not fast on all platforms
- RC4 broken by Fluhrer, Mantin, and Shamir in 2001
- More broken by Klein in 2005
- Broken even more by AlFardan, Bernstein, Paterson, Poettering, Schuldt in 2013
- . . . and even more by Garman, Paterson, and van der Merwe in 2015
- . . . and by Vanhoef and Piessens in 2015

From RC4 to eSTREAM



- Historically: two widely deployed stream ciphers:
  - AES in streaming mode (AES-CTR)
  - RC4
- Block cipher AES problematic in software implementations
  - Table-based approach has “textbook” cache-timing vulnerabilities
  - Alternative approaches (bitslicing) not fast on all platforms
- RC4 broken by Fluhrer, Mantin, and Shamir in 2001
- More broken by Klein in 2005
- Broken even more by AlFardan, Bernstein, Paterson, Poettering, Schuldt in 2013
- . . . and even more by Garman, Paterson, and van der Merwe in 2015
- . . . and by Vanhoef and Piessens in 2015
- . . . and by Bricout, Murphy, Paterson, and van der Merwe in 2016

From RC4 to eSTREAM



- Historically: two widely deployed stream ciphers:
  - AES in streaming mode (AES-CTR)
  - RC4
- Block cipher AES problematic in software implementations
  - Table-based approach has “textbook” cache-timing vulnerabilities
  - Alternative approaches (bitslicing) not fast on all platforms
- RC4 broken by Fluhrer, Mantin, and Shamir in 2001
- More broken by Klein in 2005
- Broken even more by AlFardan, Bernstein, Paterson, Poettering, Schuldt in 2013
- . . . and even more by Garman, Paterson, and van der Merwe in 2015
- . . . and by Vanhoef and Piessens in 2015
- . . . and by Bricout, Murphy, Paterson, and van der Merwe in 2016
- eSTREAM: stream-cipher competition by ECRYPT running 2004–2008

ChaCha20



- Stream cipher proposed by Bernstein in 2008
- Variant of eSTREAM finalist Salsa20
- ARX design: **A**dd, **R**otate, **X**OR
- “Easily” implementable in constant-time
- Used in TLS, SSH, and other modern protocols
- Details: `https://datatracker.ietf.org/doc/html/rfc8439`

ChaCha20



- Stream cipher proposed by Bernstein in 2008
- Variant of eSTREAM finalist Salsa20
- ARX design: **A**dd, **R**otate, **X**OR
- “Easily” implementable in constant-time
- Used in TLS, SSH, and other modern protocols
- Details: `https://datatracker.ietf.org/doc/html/rfc8439`

Parameters:

- 256-bit key (32 bytes)
- 96-bit nonce (12 bytes)
- 32-bit block counter (4 bytes)
- Generates 64-byte keystream blocks

ChaCha20

- Has a state of 64 bytes, 4×4 matrix of 32-bit words
- With c = constant, k = key, b = blockcount, and n = nonce:

cccccccc 0	cccccccc 1	cccccccc 2	cccccccc 3
kkkkkkkk 4	kkkkkkkk 5	kkkkkkkk 6	kkkkkkkk 7
kkkkkkkk 8	kkkkkkkk 9	kkkkkkkk 10	kkkkkkkk 11
bbbbbbbb 12	nnnnnnnn 13	nnnnnnnn 14	nnnnnnnn 15

Constants: `0x61707865`, `0x3320646e`, `0x79622d32`, `0x6b206574`

ChaCha20: Quarterround

- Elementary operation: *Quarterround (QR)*
  - Four add-xor-rotate operations each:
    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;`
  - Note: Addition $\mod 2^{32}$

function ColumnRound(*state*)
    QR(*state*, 0, 4, 8, 12)
    QR(*state*, 1, 5, 9, 13)
    QR(*state*, 2, 6, 10, 14)
    QR(*state*, 3, 7, 11, 15)
end function

function DiagonalRound(*state*)
    QR(*state*, 0, 5, 10, 15)
    QR(*state*, 1, 6, 11, 12)
    QR(*state*, 2, 7, 8, 13)
    QR(*state*, 3, 4, 9, 14)
end function

ChaCha20: Block Function Pseudocode




```
function ChaCha20Block(key, counter, nonce)
    state ← constants || key|| counter || nonce      ▷ In our framework, you receive state as input already
    working_state ← state
    for i = 1 to 10 do                               ▷ 10 double rounds
        ColumnRound(working_state)
        DiagonalRound(working_state)
    end for
    state ← state + working_state                    ▷ Addition mod 2^32
    return Serialize(state)
end function
```

ChaCha20: Encryption Pseudocode

function ChaCha20Encrypt(*key*, *counter*, *nonce*, *plaintext*)
    encrypted_message $\leftarrow \epsilon$ ▷ In our framework, the encryption is in-place
    for $j = 0$ **to** $\lfloor len(plaintext)/64 \rfloor - 1$ **do**
        key_stream $\leftarrow$ ChaCha20Block(*key*, *counter*+j, *nonce*)
        block $\leftarrow$ *plaintext*$[j \cdot 64 \,..\, j \cdot 64 + 63]$
        encrypted_message $\leftarrow$ *encrypted_message* $||$ (*block* $\oplus$ *key_stream*)
    end for
    if $len(plaintext) \bmod 64 \neq 0$ **then**
        $j \leftarrow \lfloor len(plaintext)/64 \rfloor$
        key_stream $\leftarrow$ ChaCha20Block(*key*, *counter*+j, *nonce*)
        block $\leftarrow$ *plaintext*$[j \cdot 64 \,..\, len(plaintext) - 1]$
        remainder $\leftarrow len(plaintext) \bmod 64$
        encrypted_message $\leftarrow$ *encrypted_message* $||$ (*block* $\oplus$ *key_stream*)$[0..remainder]$
    end if
    return *encrypted_message*
end function

ChaCha20: Ultimate Goal




```
export fn crypto_stream_chacha20_ietf_xor(
  reg u32 ct_ptr, reg ptr u8[12] nonce_ptr,
  reg ptr u8[32] sk_ptr, reg u32 ct_len)
```

Implementing ChaCha20 on Cortex-M4



- Strategy for implementing
  - Write `quarterround` function in Jasmin
  - Write `column_round` and `diagonal_round` function in Jasmin
  - Implement loop over 20 rounds in Jasmin (`chacha20_block`)
  - Implement loop over message length in Jasmin (`chacha20_encrypt_bytes`)
- Strategy for optimizing
  - Merge 4 `quarterround` functions into a full round (at least)
  - Optimize inner loop over 20 rounds
  - Keep data in registers as much as possible (reduce loads/stores)
  - Eliminate explicit `ROR` instructions

Useful features of the M4

- 16 state words won't fit into registers, you need the stack
  - `-auto-spill(-all)`
  - Use `() = #spill(X);` and `() = #unspill(X);`

Useful features of the M4

- 16 state words won't fit into registers, you need the stack
  - `-auto-spill(-all)`
  - Use `() = #spill(X);` and `() = #unspill(X);`
- Second input of arithmetic instructions goes through barrel shifter
- Can shift/rotate one input **for free**
- Examples:
  - `a = b ^ (c << 2)`: left-shift `c` by 2, xor to `b`, store result in `a`
  - `c = a + (b >>r 5)`: right-rotate `b` by 5, add to `a`, store result in `c`
  - Note: Ordering is important! Can only shift second input argument