CRYPTREC-EX-2601-2016

Security Analysis of ChaCha20-Poly1305 AEAD

KDDI Research, Inc.

February 2017

Contents

Chapter 1

Executive Summary

This report presents a security evaluation of the ChaCha20-Poly1305 Authenticated Encryption with Associated Data (AEAD). The combination of ChaCha and Poly1305 is proved to be secure authenticated encryption scheme assuming ChaCha and Poly1305 are secure algorithms. Additionally, Poly1305 is proven to be ε-almost-Δ-universal i.e., a secure universal hash function. We thus evaluate the security of ChaCha against existing attacks.

We showed that no efficient differential analysis, linear cryptanalysis and distinguish attack, guess and determine analysis, algebraic attack, and attacks on initialization process exist against ChaCha. Time-Memory-Data tradeoff attack and side-channel attack apply to ChaCha; however, we can deal with these attacks with practical countermeasures. We thus concluded that we can identify no weaknesses in ChaCha20-Poly1305 AEAD.

Chapter 2

Algorithm Description

We describe the algorithm of ChaCha, Poly1305, and ChaCha20-Poly1305 AEAD.

2.1 ChaCha

ChaCha [NL15] operates on 32-bit words, takes as input a 256-bit key $K = (k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7)$ and a 32-bit counter $C = (c_0)$, and produces a sequence of 512-bit keystream blocks[1]. This function acts on the 4×4 matrix of 32-bit words written as;

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix}$$
$$= \begin{pmatrix} \sigma_0 & \sigma_1 & \sigma_2 & \sigma_3 \\ k_0 & k_1 & k_2 & k_3 \\ k_4 & k_5 & k_6 & k_7 \\ c_0 & n_0 & n_1 & n_2 \end{pmatrix}$$

The σ and τ are constants where

$$(\tau_0, \tau_1, \tau_2, \tau_3) = (\texttt{0x61707865}, \texttt{0x3120646E}, \texttt{0x79622D36}, \texttt{0x6B206574})$$

and

$$(\sigma_0, \sigma_1, \sigma_2, \sigma_3) = (\texttt{0x61707865}, \texttt{0x3320646E}, \texttt{0x79622D32}, \texttt{0x6B206574}).$$

The keystream block Z is defined as; $Z = X + X^{(20)}$, where $X^{(r)} = \mathsf{Round}^r(X)$ with the round function of ChaCha and $+$ is word-wise addition modulo 2^{32}. If $Z = X + X^{(r)}$, it is called "r-round ChaCha" or "ChaCha$r$". The round function consists the following non-linear operations that are called

[1]The original version of ChaCha [Ber08a] suport both 128-bit and 256-bit key. However, 128-bit key is out of scope in the IETF version [NL15]. The nonce and counter have 64-bit length and they are stored (x_{12}, x_{13}) and (x_{14}, x_{15}) respectively in the original version of ChaCha.

Algorithm 1 ChaCha

```
Input: Key $K$, Counter $C$, and Nonce $N$
Output: Keystream $Z$
  Generate initial matrix $X$ using $K$, $C$, and $N$
  $y \leftarrow X$
  for $i \leftarrow 0$ to $9$ do
    /* Column Round */
    $(x_0, x_4, x_8, x_{12}) \leftarrow quarterround(x_0, x_4, x_8, x_{12})$
    $(x_5, x_9, x_{13}, x_1) \leftarrow quarterround(x_5, x_9, x_{13}, x_1)$
    $(x_{10}, x_{14}, x_2, x_6) \leftarrow quarterround(x_{10}, x_{14}, x_2, x_6)$
    $(x_{15}, x_3, x_7, x_{11}) \leftarrow quarterround(x_{15}, x_3, x_7, x_{11})$
    /* Diagonal Round */
    $(x_0, x_5, x_{10}, x_{15}) \leftarrow quarterround(x_0, x_5, x_{10}, x_{15})$
    $(x_1, x_6, x_{11}, x_{12}) \leftarrow quarterround(x_1, x_6, x_{11}, x_{12})$
    $(x_2, x_7, x_8, x_{13}) \leftarrow quarterround(x_2, x_7, x_8, x_{13})$
    $(x_3, x_4, x_9, x_{14}) \leftarrow quarterround(x_3, x_4, x_9, x_{14})$
  end for
  $Z \leftarrow X + y$
  return Z
```

quarter round functions. A vector (a, b, c, d) of four words is transformed as;

$$\begin{aligned}
a &= a + b \\
d &= d \oplus a \\
d &= (d)_{\lll 16} \\
c &= c + d \\
b &= b \oplus c \\
b &= (b)_{\lll 12} \\
a &= a + b \\
d &= d \oplus a \\
d &= (d)_{\lll 8} \\
c &= c + d \\
b &= b \oplus c \\
b &= (b)_{\lll 7}
\end{aligned}$$

The quarter-round functions are applied to the columns (x_0, x_4, x_8, x_{12}), (x_5, x_9, x_{13}, x_1), $(x_{10}, x_{14}, x_2, x_6)$ and $(x_{15}, x_3, x_7, x_{11})$ in odd round, and diagonals $(x_0, x_5, x_{10}, x_{15})$, $(x_1, x_6, x_{11}, x_{12})$, (x_2, x_7, x_8, x_{13}) and (x_3, x_4, x_9, x_{14}) in even rounds. Algorithm 1 describes the complete procedure of ChaCha.

2.2 Poly1305

Poly1305 [Ber05b] is a cryptographic message authentication code (MAC) proposed by Bernstein. The input to Poly1305 is a 256-bit one-time key and an arbitrary-length message. The output is a 128-bit tag. Algorithm 2 shows the detailed description of Poly1305. The input key has 256-bit length and is di-

Algorithm 2 Poly1305

```
Input: Key $K$ and Message $M$
Output: Tag $T$
  $(m[0], m[1], \ldots, m[d-1]) \overset{16}{\leftarrow} M$
  $d \leftarrow \lceil len(M)/16 \rceil$
  $(r, s) \overset{16}{\leftarrow} K$
  $r \leftarrow r \& \texttt{0x0FFFFFFC0FFFFFFC0FFFFFFC0FFFFFFF}$
  for $i \leftarrow 0$ to $d-1$ do
    $m[i] \leftarrow m[i] + 2^{8len(m[i])}$
  end for
  $T \leftarrow m[0]$
  for $i \leftarrow 1$ to $d-1$ do
    $T \leftarrow (r \cdot T + m[i]) \mod (2^{130} - 5)$
  end for
  $T \leftarrow (T + s) \mod 2^{128}$
  return $T$
```

Algorithm 3 ChaCha20-Poly1305 AEAD

```
Input: Key $K$, Nonce $N$, Authentication data $A$, and Message $M$
Output: Ciphertext $C$ and Tag $T$
  $z \leftarrow$ CC-Poly-KS$(K, N, len(M))$
  $C \leftarrow M \oplus z$
  $T \leftarrow$ CC-Poly-T$(K, N, A, C)$
  return $(C, T)$
```

vided to two 128-bit keys r and s. The algorithm clamps 22-bits of r. The output tag is $((m[0]r^n + m[1]r^{n-1} + \cdots + m[n]) \mod (2^{130} - 5) + s) \mod 2^{128}$, where $M[i]$ is i-th 16-bit chunk of the input message M.

2.3 ChaCha20-Poly1305 AEAD

ChaCha20-Poly1305 is an authenticated encryption with additional data algorithm. The input data are a 256-bit key K; a 96-bit nonce N; arbitrary length authenticated data A, and an arbitrary length message M. Algorithm 3 describes the complete procedure of ChaCha20-Poly1305 AEAD and Fig. 2.1 shows the overall structure. Algorithm 4 and 5 show its subroutines.

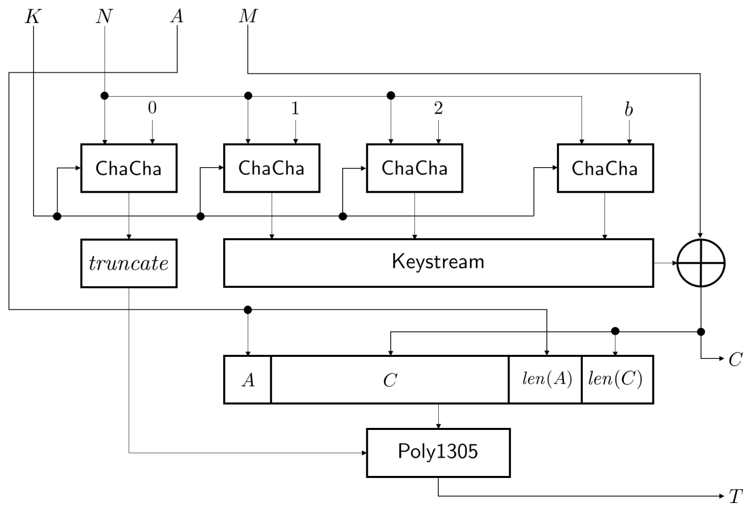

Figure 2.1: ChaCha20-Poly1305 AEAD

Algorithm 4 CC-Poly-KS

```
Input: Key $K$, Nonce $N$ and Input length $L$
Output: Keystream $Z$
  $b \leftarrow \lceil L/64 \rceil$
  for $i \leftarrow 0$ to $b-1$ do
    $z[i] \leftarrow \mathsf{ChaCha}(K, i+1, N)$
  end for
  $z \leftarrow \sum_{i=0}^{b-1} z[i] \cdot 2^{512i}$
  $Z \leftarrow \mathit{trancate}(l, z)$
  return $Z$
```

Algorithm 5 CC-Poly-T

```
Input: Key $K$, Nonce $N$, Authentication data $A$, and Message $M$
Output: Tag $T$
  $k \leftarrow \mathit{trancate}(32, \mathsf{ChaCha}(K, 0, N))$
  $y \leftarrow A$
  $y \leftarrow y + M \cdot 2^{128\lceil len(A)/16 \rceil}$
  $y \leftarrow y + len(A) \cdot 2^{128(\lceil len(A)/16 \rceil + \lceil len(M)/16 \rceil)}$
  $y \leftarrow y + len(M) \cdot 2^{128(\lceil len(A)/16 \rceil + \lceil len(M)/16 \rceil + 1/4)}$
  $T \leftarrow \mathsf{Poly1305}(k, y)$
  return $T$
```

Chapter 3

Security Analyses

3.1 Related Work

Crowley [Cro06] presented a differential cryptanalysis on reduced round Salsa20/5 that uses a 3-round differential and requires 2^{165} time complexity and work and 2^{6} plaintexts. Fischer et al. [FMB^{+}06] exploited a 4-round differential to attack Salsa20/6 within 2^{177} time complexity. Tsunoo et al. [TSK^{+}07] attacked Salsa20/7 within 2^{190} time complexity using a 4-round differential, and broke Salsa20/8 with 2^{255} time complexity. Aumasson et al. [AFK^{+}08] reduced the time complexity to 2^{151} for Salsa20/7 and 2^{251}. for Salsa20/8 still using a 4-round differential. They also presented that ChaCha6 and ChaCha7 can be attacked with time complexity 2^{139} and 2^{248}, respectively. Shi et al. [SZFW12] proposed improved attack based on second-order differential with 2^{148} time complexity for Salsa20/7, 2^{250}. for Salsa20/8, 2^{136} for ChaCha6 and $2^{246.5}$ for ChaCha7. Maitra [Mai16] showed a chosen IV cryptoanalysis and the time complexity of the attack can be reduced to $2^{245.5}$ for Salsa20/8 and 2^{239} for ChaCha7. Mouha and Preneel [MP13] proposed a method to search for optimal differential characteristics for ARX ciphers and applied it to find characteristics Salsa20/3. Choudhuri and Maitra [CM16] evaluate the security of Salsa and ChaCha against differential cryptanalysis using a hybrid model of non-linear round functions and linear approximation. The summary of existing attacks is shown in Table 3.1. They concluded that Salsa20/12 and ChaCha12 are sufficient for 256-bit keys under the attack model.

Procter [Pro14] demonstrated that the combination of ChaCha and Poly1305 is a secure authenticated encryption scheme assuming ChaCha is a pseudo-random function (PRF), and Poly1305 is ε-almost-Δ-universal. Imamura and Iwata [KI16] show key-recovery attack and forge attack are possible in a nonce-misuse situation where the same nonce is repeatedly used.

3.2 Analysis on ChaCha Structure

ChaCha is similar not to conventional stream cipher algorithms but ARX-type block ciphers. The structure of ChaCha should be understood as block cipher algorithm and analyze it focusing on the round function.

3.2.1 Invertibility

The quarter-round function of ChaCha contains only additions, exclusive-or, and constant-distance rotations. These basic operations can be described as follows;

$$\begin{aligned}
\texttt{add} &: (x, y) \mapsto (x + y, y),\\
\texttt{xor} &: (x, y) \mapsto (x \oplus y, y),\\
\texttt{rotl} &: x \mapsto (x)_{\lll n},
\end{aligned}$$

and they are invertible:

$$\begin{aligned}
\texttt{add}^{-1} &: (x, y) \mapsto (x - y, y),\\
\texttt{xor}^{-1}(= \texttt{xor}) &: (x, y) \mapsto (x \oplus y, y),\\
\texttt{rotl}^{-1}(= \texttt{rotr}) &: x \mapsto (x)_{\ggg n}.
\end{aligned}$$

The quarter-round function of ChaCha is thus invertible, and the inverse quarter-round function is given as;

$$\begin{aligned}
b &= (b)_{\ggg 7}\\
b &= b \oplus c\\
c &= c - d\\
d &= (d)_{\ggg 8}\\
d &= d \oplus a\\
a &= a - b\\
b &= (b)_{\ggg 12}\\
b &= b \oplus c\\
c &= c - d\\
d &= (d)_{\ggg 16}\\
d &= d \oplus a\\
a &= a - b.
\end{aligned}$$

The round function of ChaCha consists of four quarter-round functions and distinct 4-tuples of words are processed with the quarter-round functions. Thus, the round function is also invertible. There is no entropy loss in the process of the round function.

Note that the entire key-generation process is not necessarily invertible. The keystream Z is calculated as $X + X^{(20)}$ and irreversible addition $\texttt{irradd} : (x, y) \mapsto (x + y)$ is used.

3.2.2 Structual difference from Salsa20

The Salsa20 algorithm is described as follows;

Sala20 Algorithm The stream cipher Salsa20 [Ber08b] operates on 32-bit words, takes as input a 256-bit key $K = (k_0, k_1, k_2, k_3, k_4, k_5, k_6, k_7)$ or 128-bit key $K = (k_0, k_1, k_2, k_3)$ and a 64-bit nonce $N = (n_0, n_1)$ and counter $C =$

(c_0, c_1), and produces a sequence of 512-bit keystream blocks. This function acts on the 4×4 matrix of 32-bit words written as;

$$X = \begin{pmatrix} x_0 & x_1 & x_2 & x_3 \\ x_4 & x_5 & x_6 & x_7 \\ x_8 & x_9 & x_{10} & x_{11} \\ x_{12} & x_{13} & x_{14} & x_{15} \end{pmatrix}$$

$$= \begin{cases} \begin{pmatrix} \tau_0 & k_0 & k_1 & k_2 \\ k_3 & \tau_1 & n_0 & n_1 \\ c_0 & c_1 & \tau_2 & k_4 \\ k_5 & k_6 & k_7 & \tau_3 \end{pmatrix} & (K \text{ is a 128-bit key}) \\ \begin{pmatrix} \sigma_0 & k_0 & k_1 & k_2 \\ k_3 & \sigma_1 & n_0 & n_1 \\ c_0 & c_1 & \sigma_2 & k_0 \\ k_1 & k_2 & k_3 & \sigma_3 \end{pmatrix} & (K \text{ is a 256-bit key}) \end{cases}$$

The σ and τ are constants where

$$(\tau_0, \tau_1, \tau_2, \tau_3) = (\texttt{0x61707865}, \texttt{0x3120646E}, \texttt{0x79622D36}, \texttt{0x6B206574})$$

and

$$(\sigma_0, \sigma_1, \sigma_2, \sigma_3) = (\texttt{0x61707865}, \texttt{0x3320646E}, \texttt{0x79622D32}, \texttt{0x6B206574}).$$

The keystream block Z is defined as; $Z = X + X^{(20)}$, where $X^{(r)} = \mathsf{Round}^r(X)$ with the round function of Salsa20 and $+$ is word-wise addition modulo 2^{32}. If $Z = X + X^{(r)}$, it is called "r-round Salsa20" or "Salsa20/r". The round function consists the following nonlinear operations that are called quarter round functions. A vector (a, b, c, d) of four words is transformed as;

$$\begin{aligned} b &= b \oplus ((d + a)_{\lll 7}) \\ c &= c \oplus ((a + b)_{\lll 9}) \\ d &= d \oplus ((b + c)_{\lll 13}) \\ a &= a \oplus ((c + d)_{\lll 18}) \end{aligned}$$

The quaterround function are applied to columns (x_0, x_4, x_8, x_{12}), (x_5, x_9, x_{13}, x_1), $(x_{10}, x_{14}, x_2, x_6)$ and $(x_{15}, x_3, x_7, x_{11})$ in odd round, and rows (x_0, x_1, x_2, x_3), (x_5, x_6, x_7, x_4), $(x_{10}, x_{11}, x_8, x_9)$ and $(x_{15}, x_{12}, x_{13}, x_{14})$ in even rounds. Algorithm 6 describes the complete procedure of Salsa20.

Deference from Salsa20 The quarter-round function of ChaCha contains four additions, four exclusive-ors and four constant-distance rotations of 32-bit words, similar to that of Salsa20. However, ChaCha increases the amount of diffusion in the quarter-round functions comparing with Salsa20. Figure 3.1 and 3.2 shows the quarter round function of ChaCha and Salsa20. Each input word is updated twice in ChaCha quarter-round function whereas each word is update once in Salsa20 quarter-round function. Furthermore, ChaCha quarter-round function gives each input word a chance to affect each output word similar to Salsa20 quarter-round function.

Algorithm 6 Salsa20

```
Input: Key $K$, Counter $C$, and Nonce $N$
Output: Keystream $Z$
  Generate initial matrix $X$ using $K$, $C$, and $N$
  $y \leftarrow X$
  for $i \leftarrow 0$ to $9$ do
    /* Column Round */
    $(x_0, x_4, x_8, x_{12}) \leftarrow quarterround(x_0, x_4, x_8, x_{12})$
    $(x_5, x_9, x_{13}, x_1) \leftarrow quarterround(x_5, x_9, x_{13}, x_1)$
    $(x_{10}, x_{14}, x_2, x_6) \leftarrow quarterround(x_{10}, x_{14}, x_2, x_6)$
    $(x_{15}, x_3, x_7, x_{11}) \leftarrow quarterround(x_{15}, x_3, x_7, x_{11})$
    /* Row Round */
    $(x_0, x_1, x_2, x_3) \leftarrow quarterround(x_0, x_1, x_2, x_3)$
    $(x_5, x_6, x_7, x_4) \leftarrow quarterround(x_5, x_6, x_7, x_4)$
    $(x_{10}, x_{11}, x_8, x_9) \leftarrow quarterround(x_{10}, x_{11}, x_8, x_9)$
    $(x_{15}, x_{12}, x_{13}, x_{14}) \leftarrow quarterround(x_{15}, x_{12}, x_{13}, x_{14})$
  end for
  $Z \leftarrow X + y$
  return Z
```

The rotation distance in the quarter-round function is 16, 12, 8, 7 in ChaCha and 7, 9, 13, 18 in Salsa20. There is no report on security difference due to the rotation distances. Three distances can be divided by 4 and two can be divided by 8 in ChaCha. The change of rotations distance may contribute to improving the performance in embedded environments with 8-bit or 4-bit CPU.

3.3 Cryptanalysis on ChaCha

3.3.1 Differential Analysis

Differential cryptanalysis introduced by Biham and Shamir [BS93] is now used as a general method for analyzing various cryptographic primitives including stream ciphers. Currently, almost all cryptanalysis of Salsa20 and Chacha are based on the technique of differential analysis.

The most general idea of differential attacks is to exploit pairs of plaintexts with certain differences which yield other certain differences in the corresponding ciphertexts (or any internal states of the cipher) with a non-uniform distribution. Specifically, consider a system with input an n-bit string $X = [X_1, X_2, \ldots, X_n]$ and output an n-bit string $Y = [Y_1, Y_2, \ldots, Y_n]$. Denote a pair of input as (X', X'') and the corresponding pair of output as (Y', Y''), respectively. The input difference is denoted by $\Delta X = X' \oplus X''$ where "$\oplus$" may represent any binary operation on n-bit strings which defines the difference between the two operands and is, in its most general form, the bitwise Exclusive-OR operation. The difference between X' and X'' is $\Delta X = [\Delta X_1, \Delta X_2, \ldots, \Delta X_n]$ where $\Delta X_i = X'_i \oplus X''_i$ with X'_i and X''_i being the i-th bit of X' and X'', respectively. Similarly, $\Delta Y = Y' \oplus Y'' = [\Delta Y_1, \Delta Y_2, \ldots, \Delta Y_n]$ is the output difference. Differential cryptanalysis tries to identify a scenario where a particular ΔY occurs given a particular ΔX with a probability differing much from uniform. Note

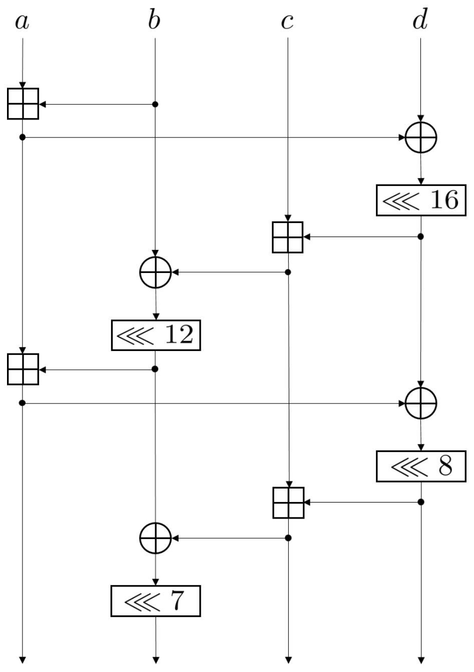

Figure 3.1: Quarter-round function of ChaCha

that the probability $\Pr[\Delta Y|\Delta X]$ can be either much greater than or less than the uniform probability. When such ΔY exists, we call the pair $(\Delta X, \Delta Y)$ a *differential*. Such a differential can be used either to distinguish the ciphertext from randomness or to help recover the key used by the cipher.

To recover the (partial) key of the cipher, we work backward from an output by guessing a part of secret key (we call subkey from now on). More specifically we feed the cipher with many chosen plaintexts consisting of pairs of plaintexts with a difference ΔX to generate the ciphertexts C and try to decrypt these ciphertexts using all possible subkeys to get the internal states Y. By checking the frequency that ΔY occurs, we can select the correct subkey with high probability. If our guess of the subkey is not the correct one, then the frequency of ΔY equals the probability of ΔY in the differential analysis is highly impossible. But for the correct guess, we will observe a frequency of ΔY quite close to its conjectured value $\Pr[\Delta Y|\Delta X]$.

Truncated differential analysis In the above discussion we treat the output difference ΔY as an n-bit string. But as shown by Knudsen [Knu95]it is not always necessary to predict the full n bit value of the output difference. Sometimes even a 1-bit value in the output difference suffices for the cryptanalysis. Such kind of differential which only predicts parts of an n-bit value is called a *truncated differential*.

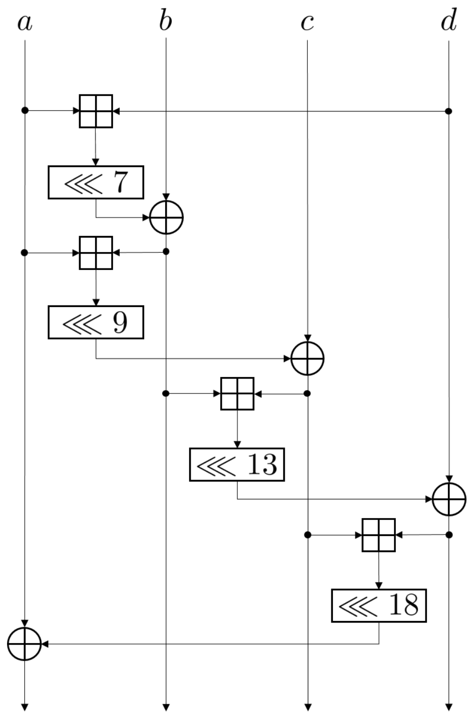

Figure 3.2: Quarter-round function of Salsa20

Cryptanalysis of Salsa20

Crowley [Cro06] presented the first attack of 256-bit Salsa20/5 using the technique of truncated differential analysis in 2005. Crowley identified a differential which as a biased bit in the output difference in the third round of Salsa20 core function. The attack only uses chosen IV, i.e, the nonce (n_0, n_1) and the counter (c_0, c_1). The truncated differential $(\Delta X, \Delta Y)$ Crowley has found is the following one;

$$\Delta X = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \texttt{0x80000000} & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{3rounds}} \Delta Y = \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & * \\ \texttt{0x02002802} & * & * & * \end{pmatrix},$$

with a theoretic probability 2^{-12}. But In his experiments Crowley observed a much higher probability of about 2^{-9} for this truncated probability in practice. Using this differential Crowley amounted a key recover attack which works two rounds backwards from a ciphertext by guessing 160 relevant key bits. This attack requires 2^{165} work and 2^6 keystreams.

Fischer et al. [FMB+06] reported an attack on 256-bit Salsa20/6. This attack used a similar technique of truncated differential analysis as compared with Crowley. But Fischer et al. has found a differential with a biased bit in the output difference in the fourth round which helped to push the attack against a further round.

To find a differential path for the four rounds Salsa, Fischer et al. first introduced an alternative LinSalsa20 primitive which replaces the addition modulo 2^{32} in Salsa20 with bitwise XOR and then analyzed some low-weight differential path for LinSalsa20. Under the condition that addition does not yield a carry, LinSalsa20 behaves the same as the actual Salsa 20. Using this observation they successfully applied the differential path found in LinSalsa20 to Salsa20 and mounted a key recovery attack on Salsa20/6. They also observed non-randomness for Salsa until round 7. Their attack on 256-bit Salsa20/6 used a differential found after four rounds operation of Salsa20, and work two rounds backward by guessing 160 relevant key bits. This attack requires 2^{177} computational complexity and 2^{16} keystreams.

Tsunoo et al. [TSK+07] used the characteristics of differential of modulo addition [LM01] to rigorously compute the differential probabilities for each bit in the round 4 internal state of Salsa20. They found the following two 4-round differentials to have the most biased probability of approximately $(1-2^{-5.24})/2$;

$$\Delta X_1 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \texttt{0x80000000} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{4rounds}} \Delta Y_1 = \begin{pmatrix} * & \texttt{0x00400000} & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix},$$

$$\Delta X_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ \texttt{0x80000000} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{4rounds} \Delta Y_2 = \begin{pmatrix} * & * & * & * \\ * & * & \texttt{0x00400000} & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}.$$

Then using these differentials they reported a key recovery attack by working 3 rounds backwards from Salsa20/7. They used nonlinear approximation of integer addition to reduce the guessing space for relative key bits to 171. Their attack requires 2^{184} work and $2^{11.4}$ keystreams. They also reported a 2^{255} complexity attack using 2^{10} keystreams with probability 44% to narrow down the correct key of Salsa20/8. However, this can hardly be considered as a useful attack because it is effectively slower than the brute force attack. Since exhaustive search succeeds with probability 50% within the same number of trials, with much less data and no additional computations [AFK+08].

Aumasson et al. [AFK+08] proposed attacks on Slasa20/8 and Salsa20/7 and Chacha6. Their attack used the notion of probabilistic neutral bits (PNB) introduced by Biham and Chen [BC04] to split key bits into two subsets as the relevant key bits which can be filtered by observations of a biased output differential and the less significant key bits which are determined by exhaustive search. Their attacks on Salsa20 worked four rounds forwards to find the biased bit in the differential and.

Aumasson et al.'s attack on 256-bit Salsa20/7 uses the following 4-round differential with a median bias $|\varepsilon_d^*| = 0.131$.

$$\Delta X = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \texttt{0x00010000} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{4rounds}} \Delta Y = \begin{pmatrix} * & \texttt{0x02000000} & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}.$$

Using this differential the attack worked 3 rounds backwards by guessing This attack requires 2^{151} work and 2^{26} keystreams. Their attack on 128-bit Salsa20/7 used the same 4-round differential and broke Salsa20/7 within 2^{111} time and 2^{21} keystreams. The same 4-round differential was used to attack Salsa20/8 by working four rounds backward. This attack requires 2^{251}-operation and 2^{31} keystreams.

Following the work of Aumasson et al., Shi et al. [SZFW12] consider a new type of distinguisher called column (row) chaining distinguishers (CCD) which can efficiently make use of the biases of multiple differential trails and the matrix structure of the cipher. Besides, they found new high probability second-order differential trails that were not covered by previous results. They also generalized the notion of PNB to probabilistic neutral vectors (PNV) which explore the properties against more than one flipped input bit. These new techniques and notions can effectively improve both time and data complexity of previous attacks against both Salsa and ChaCha. For example, one of the highly biased 4-round second-order differential of Salsa is as follows;

$$\Delta X = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \texttt{0x00080000} \\ \texttt{0x00400000} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\xrightarrow{\text{4rounds}} \Delta Y = \begin{pmatrix} * & \texttt{0x10000000} & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}.$$

Their attack on 256-bit Salsa20/5 used five 3-round differentials by working two rounds backward from a 5-round keystream block. This attack can break Salsa20/5 within time 2^{55} and data 2^{10}. Their attack on 256-bit Salsa20/6 used four 4-round differentials as the chaining distinguisher by working two rounds backward from Salsa20/6 and requires 2^{73} operations and 2^{16} keystreams. To attack 256-bit Salsa20/7, they used the same 4-round differential as that of Aumasson et al. This attack requires 2^{148} operations and 2^{24} keystreams. The same differential was used to attack 256-bit Salsa20/8 with a time complexity of 2^{250} and data complexity of 2^{27}. Finally, they showed an attack on 128-bit Salsa20/7 using the same 4-round differential which requires 2^{109} operations and 2^{19} keystreams.

In 2015, Maitra[MPM15, Mai16] improved the complexity of attacking Salsa and ChaCha. In [MPM15], Maitra revisited the single bit differentials for 4-round Salsa and found better biases which are more significant than previously published ones. He also revisited technique of PNB used in Aumasson et al. [AFK$^+$08] and observed that in practice the median of certain biases are 4 times more than what was observed by Aumasson. Maitra further explored the tradeoff between carefully choosing more PNBs at the cost of accepting less probability for distinguishing the correct key from the wrong keys. A combination of the above process resulted in an attack on 256-bit Salsa20/8 with $2^{247.2}$ work and 2^{27} keystreams. In a subsequent work [Mai16], Maitra reported chosen IV cryptanalysis on reduced round Salsa and ChaCha. In [Mai16], Maitra follows his previous idea of exploring more PNBs to obtain better results for the cryp tanalysis of Salsa and Chacha. He also showed how to exploit specific IVs (the nonces and counters) corresponding to the secret key which can help improve the attack of Aumasson et al. [AFK$^+$08]. His attack on 256-bit Salsa20/8 used the following 4-round differential by working 4 rounds backwards.

$$\Delta X = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \texttt{0x0x00010000} \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \xrightarrow{\text{4rounds}} \Delta Y = \begin{pmatrix} * & \texttt{0x02000000} & * & * \\ * & * & * & * \\ * & * & * & * \\ * & * & * & * \end{pmatrix}.$$

The attacks requires $2^{245.5}$ work and $2^{22.5}$ keystreams.

Cryptanalysis on ChaCha

Due to the similar design of ChaCha and Salsa20, the cryptanalysis on Salsa discussed above also apply to ChaCha. However, since ChaCha are designed to achieve a quick diffusion, it is suggested harder to break than Salsa. This is evidenced by the published attacks on Salsa and ChaCha.

The initial attack on ChaCha was started by Aumasson et al. [AFK+08] using the technique of probabilistic neutral bits (PNB). Aumasson et al. used the following 3-round differential to attack ChaCha6 and ChaCha7:

$$\Delta X = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & \texttt{0x04000000} & 0 & 0 \end{pmatrix} \xrightarrow{\text{3rounds}} \Delta Y = \begin{pmatrix} * & * & * & * \\ * & * & * & * \\ * & * & * & \texttt{0x80000000} \\ * & * & * & * \end{pmatrix}.$$

The attack on 256-bit ChaCha6 worked three rounds backward from the 6-round keystream block and required 2^{139} operations and 2^{30} keystreams. By working four rounds backwards from ChaCha7, they could successfully attack 256-bit ChaCha7 using time 2^{248} and data 2^{27}. Same experiments were done on the 128-bit variants. Their attack on 128-bit ChaCha6 can break it within 2^{107} time and 2^{30} keystreams. However, their method failed on 128-bit ChaCha7.

The tools of Shi et al. [SZFW12], column chaining distinguisher (CCD) and probabilistic neutral vectors (PNV) also applies to the analysis of reduced ChaCha. They used the same 3-round differential as that of Aumasson et al. [AFK+08] to attack ChaCha6 and ChaCha7. Due to the generalized techniques and notions compared with [AFK+08], their attacks have better complexity. Specifically for 256-bit ChaCha6, their attack requires 2^{136} operations and 2^{28} keystreams; their attack can break 256-bit ChaCha7 within time $2^{246.5}$ and data 2^{27}, and their attack on 128-bit ChaCha6 uses 2^{105} work and 2^{28} keystreams. They did not report results on attacks for 128-bit ChaCha7.

The latest improvement on the attacks against reducing ChaCha came from Maitra [Mai16] in 2015. The attack used the same tools for attacking Salsa20 such as exploring more PNBs to obtain better results and exploit specific IVs corresponding to the secret key. Maitra improved the complexity of the attack on 256-bit ChaCha7 to 2^{239} operations and 2^{24} keystreams.

Differential-Linear Cryptanalysis

Differential-linear cryptanalysis discovered by Langford and Hellman [LH94] uses the idea to combine the techniques of differential cryptanalysis and linear analysis to derive bias for the underlying cipher up to more rounds. It usually divides the cipher E into two parts $E = E_1 \circ E_0$. Differential-linear cryptanalysis uses a differential $\delta_i \xrightarrow{E_0} \delta_o$ and a linear approximation $\chi_i \xrightarrow{E_1} \chi_o$ and build trials or characteristics for both half individually, where δ_i (δ_o) is the input (output) difference for the first half of the cipher E_0 and χ_i (χ_o) is the input (output) mask for the second half of the cipher E_1, respectively.

Recently Choudhuri and Maitra [CM16] developed a differential-linear cryptanalysis on reduced round Salsa20 and ChaCha. This is known as currently the most effective attack on reduced round Salsa20 and ChaCha published so far. The improvement of their attack mainly comes from the idea of using multi-bit differentials instead of the single-bit truncated differentials in existing works. By considering the linear combination of multi-bit in the output difference, Choudhuri and Maitra have successfully identified high biases in Salsa20 after six rounds and ChaCha after five rounds, which demonstrates a two rounds of improvement for both the ciphers over previous cryptanalytic results. Their results of attacks on 256-bit Salsa20 and ChaCha can be listed as follows:

- 5-round Salsa20: time complexity 2^8 and data complexity 2^8.
- 6-round Salsa20: time complexity 2^{32} and data complexity 2^{32}.
- 7-round Salsa20: time complexity 2^{137} and data complexity 2^{61}.
- 8-round Salsa20: time complexity $2^{244.9}$ and data complexity 2^{96}.
- 4-round ChaCha: time complexity 2^6 and data complexity 2^6.
- 5-round ChaCha: time complexity 2^{16} and data complexity 2^{16}.
- 6-round ChaCha: time complexity 2^{116} and data complexity 2^{116}.
- 7-round ChaCha: time complexity $2^{237.7}$ and data complexity 2^{96}.

Note: Biham and Carmeli [BC14] proposed a new technique of partitioning the data set to improve the performance of linear cryptanalysis against FEAL-8X. Recently this partitioning technique was generalized by Leurent [Leu16] to multi-bit partitioning and also to differential cryptanalysis. Leurent applied the generalized partitioning technique to the cryptanalysis of an ARX-style message authentication code, Chaskey and got significant improvement. The Choudhuri-Maitra differential-linear attack was inspired by the work of Leurent but without the partitioning technique. Therefore we assume there is room for improvement of the differential-linear cryptanalysis on ChaCha and Salsa using the partitioning technique.

Rotational Cryptanalysis

Rotational cryptanalysis [KN10, KNP+15]is a probabilistic analysis especially applicable to word oriented ciphers which use almost rotation-invariant operations. ChaCha is one example of the ARX cipher family which use only (modular) Addition, Rotation and Xor operations.

It can be viewed as a special case of differential attack. But the difference metrics is now defined by rotational difference other than the classic xor difference.

To launch a rotational attack, one starts from a rotational pair, i.e. two states X and X' where the words of X' are all rotations of the words of X

by a fixed amount. If the corresponding outputs for such rotational pair also form a rotational pair with a probability higher than in the case of a random permutation, then this bias can be utilized to distinguish the cipher from a random permutation, and it can further lead to key recovery attack.

However, one crucial requirement of rotational cryptanalysis is that the constants used in the cipher construction must be rotation-invariant, i.e. the constants must preserve their values when rotated [KNP$^+$15]. In the design of Salsa20 and ChaCha, the constants are carefully selected to be non-symmetric to prevent any use of rotational pairs [Ber05a].

Boomerang Attack

Boomerang attack [Wag99] is another differential-style attack which exploits the differential in half of the rounds to boost the bias of certain differential characteristics.

The attack divides the cipher E into two parts $E = E_1 \circ E_0$, where E_0 represents the first half of the cipher and E_1 represents the second half. In order to lunch boomerang attack one needs to find differential characteristics for the two halves of the cipher respectively. Denote $\Delta \to \Delta'$ a differential characteristic for E_0 and $\bigtriangledown \to \bigtriangledown'$ a differential characteristic for E_1^{-1}. Consider four plaintexts P, P', Q, Q' and their corresponding ciphertexts C, C', D, D'. The plaintexts and ciphertexts are generated in the following way: 1), generate $P' = P \oplus \Delta$; 2), get the encryption C, C' of P, P' with two chosen-plaintext queries; 3), generate D, D' as $D = C \oplus \bigtriangledown$ and $D' = C' \oplus \bigtriangledown$; 4), decrypt D, D' to get the corresponding plaintexts Q, Q' with two adaptive chosen-ciphertext queries. The four plaintexts P, P', Q, Q' generated in the above manner are called a quartet. The property of the quartet is that P, P' satisfy the characteristic for E_0, P, Q and P', Q' satisfy the characteristic for E_1^{-1} and Q, Q' satisfy the characteristic for E_0^{-1}, $\Delta' \to \Delta$. Then using this quartet we can distinguish the cipher from a random permutation and can further lunch key recovery attack.

But one disadvantage of the boomerang attack is that it inherently requires the ability to perform both chosen-plaintext query and adaptive chosen-ciphertext query simultaneously which may limit the application in a practical attack. Also, it is not obvious how boomerang attack can be applied to ChaCha since the construction of ChaCha uses fixed constants at pre-defined locations in the initial state. The implication is that when using an adaptive chosen-ciphertext query, the altered ciphertext may not lead to a valid plaintext. Currently, there is no report on successful boomerang attack on ChaCha or Salsa20.

Security Evaluation of Salsa20 and ChaCha

Though several attacks have been developed against reduced Salsa20 and ChaCha, to the best of our knowledge there has been no report on the weakness of full round Salsa20/20 or ChaCha. Recently Choudhuri and Maitra [CM16] evaluated the security of Salsa20 and ChaCha against differential cryptanalysis. In their analysis, they introduced the hybrid model for the evaluation of differential cryptanalysis of Salsa20 and ChaCha. In the hybrid model, the initial rounds are run using the nonlinear function as in the real cipher, but subsequent rounds are run using the linearized counterpart. To utilize this hybrid model they first examine the biases in the cipher after a few forward rounds and then estimate an upper bound on the number of rounds till such biases can be observed. Because Salsa20 and ChaCha both use only use modular addition and XOR, it is possible to upper bound the absolute values of the biases of Salsa20 and ChaCha by those of the linearized counterparts. Combining both the forward biases and backward biases, they claimed that Salsa20/12 and ChaCha12 are sufficient to provide security against certain kinds of differential cryptanalysis for 256-bit keys.

3.3.2 Linear Cryptanalysis and Distinguishing Attack

The quarter-round function of ChaCha consists of four additions. We have a linear expression of the round function by approximating these additions by exclusive-or operations.

Sarkar [Sar09] evaluated the probability of the i-th bits of $S^{(n)} = X^{(0)} + X^{(1)} + \cdots + X^{(n-1)}$ and $L^{(n)} = X^{(0)} \oplus X^{(1)} \oplus \cdots \oplus X^{(n-1)}$ are identical; that is, $\gamma_i^{(n)} = \Pr[S_i^{(n)} = L_i^{(n)}]$. The probability $\gamma_i^{(2)}$ equals to $(1 + 2^{-i})/2$ for $n = 2$.

Let x and y be integers over $GF(2^{32})$. The probability $x + y = x \oplus y$ holds is given as;

$$\Pr[x + y = x \oplus y] = \prod_{i=0}^{31} \gamma_i^{(2)} = 2^{-29.75}.$$

The quarter-round functions of ChaCha is given in section 3.2.3 and each function contain more than one addition. The maximum linear probability of the quarter-round function is thus bounded by $2^{-29.75}$. Note that equivalence between addition and exclusive-or does not always hold, and the upper bound is independent of the number of addition. We ignore the rotate operations of the quarter-round functions in this estimate. The maximum linear probability of 20-round ChaCha is bounded by $(2^{-29.75})^{20} = 2^{-595}$. We conclude that linear cryptanalyses and distinguishing attacks on ChaCha are not effective.

3.3.3 Guess and Determine Analysis

We first analyze the quarter-round function. The quarter-round function is described as follows:

$$\begin{aligned}
a_{j+1} &= (a_j + b_j) + (b_j \oplus (c_j + (d_j \oplus (a_j + b_j))_{\lll 16}))_{\lll 12}) \\
b_{j+1} &= ((b_j \oplus (c_j + (d_j \oplus (a_j + b_j))_{\lll 16}))_{\lll 12}) \\
&\quad \oplus ((c_j + (d_j \oplus (a_j + b_j))_{\lll 16}) + ((((d_j \oplus (a_j + b_j))_{\lll 16}) \\
&\quad \oplus ((a_j + b_j) + (b_j \oplus (c_j + (d_j \oplus (a_j + b_j))_{\lll 16}))_{\lll 12})))_{\lll 8}))_{\lll 7} \\
c_{j+1} &= (c_j + (d_j \oplus (a_j + b_j))_{\lll 16}) + ((((d_j \oplus (a_j + b_j))_{\lll 16}) \\
&\quad \oplus ((a_j + b_j) + (b_j \oplus (c_j + (d_j \oplus (a_j + b_j))_{\lll 16}))_{\lll 12})))_{\lll 8}) \\
d_{j+1} &= (((d_j \oplus (a_j + b_j))_{\lll 16}) \\
&\quad \oplus ((a_j + b_j) + (b_j \oplus (c_j + (d_j \oplus (a_j + b_j))_{\lll 16}))_{\lll 12})))_{\lll 8}
\end{aligned}$$

Each output value $a_{j+1}, b_{j+1}, c_{j+1}, d_{j+1}$ is calculated from all input values (a_j, b_j, c_j, d_j). If adversary obtain all output values, the adversary has to guess at least three 32-bit input values in order to obtain remaining one 32-bit input value. Even though the above guess-and-determine approach applies to one round operation using the quarter function, the adversary has to guess 3×4 32-bit values to break two rounds operation that consists of the column round operation and the diagonal round operation. Furthermore, keystream bit is calculated after adding initial state bits as $Z \leftarrow X + X^{(20)}$. Thus, a guess-and-determine approach cannot recover an initial key with reason a computational cost.

3.3.4 Time-Memory-Data Tradeoff Attack

Baic ideas of time-memory-data tradeoff attack against a stream cipher are proposed by Babbage [Bab95] and Golić [Gol97]. Birkov and Shamir [BS00] proposed an advanced scheme that combines the basic ideas and an attack against a block cipher [Hel80].

They show the tradeoff formula as follows;

$$TM^2D^2 = N^2, \quad P = N/D \quad (D^2 \leq T \leq N).$$

The five key parameters are defined as followings:

- N: the size of the search space
- P: the time required by preprocessing phase of the attack
- M: the amount of random access memory
- T: the time required by real-time phase of the attack
- D: the amount of real-time data available to the attack

Hong and Saker [HS05] evaluate the security of stream cipher with IV based on the tradeoff. The size of the search space is $N = 2^{k+v}$ where k and v are the bit length of the key and IV, respectively.

In the original version of ChaCha takes a 256-bit key and 64-bit nonce and $N = 2^{320}$. Thus, $P = N^{3/4} = 2^{240}$, $D = N^{1/4} = 2^{80}$, $M = N^{1/2} = 2^{160}$, and $T = N^{1/2} = 2^{160}$ satisfies the above the tradeoff formula, the total time complexity of the attack $P + T \approx 2^{240}$ is less than 2^{256}.

In the IFIT version of ChaCha takes a 256-bit key and 96-bit nonce and $N = 2^{352}$. A time-memory-data tradeoff attack is theoretically possible against the version of ChaCha. For example, $P = N^{2/3} = 2^{234.67}$, $D = N^{1/3} = 2^{117}$, $M = N^{1/2} = 2^{176}$, and $T = N^{1/2} = 2^{176}$ satisfies the above the tradeoff formula, the total time complexity of the attack $P + T \approx 2^{234.67}$ is less than 2^{256}. However, we can limit the amount of the real-time data up to 2^{96} to have a total time is greater than 2^{256}. Table 3.2 shows the typical parameter of the time-memory-data tradeoff attack.

3.3.5 Algebraic Attack

The algebraic degree and the number of terms in the ANF of the component Boolean functions of modular addition can be obtained as follows [BS05]:

$$
\begin{aligned}
d(f_i) &= i + 1 \\
n(f_i) &= 2^i + 1
\end{aligned}
$$

Every round has at least four 32-bit additions; thus after 20 round operations, d and n can be expected to be sufficiently large, and any algebraic attacks are not efficient for breaking ChaCha. Currently, no algebraic attack is found.

3.3.6 Attacks on Initialization Process

There is no difference between an initialization process and a keystream generation process of ChaCha. The cipher has initial key loading process, and the key, iv and counter values are loaded for every keystream generation. Thus, no specific attack on the initialization process exists on ChaCha.

3.3.7 Analysis on Period

We estimate the variety of the output keystream of ChaCha to evaluate the periodicity. ChaCha is assumed to be a random function in this analysis.

The input of the IETF version of ChaCha is a 256-bit key, a 32-bit block counter, and a 96-bit nonce; thus, the number of elements in the input set of ChaCha is 2^{384} The number of possible elements in the output set is up to 2^{384}. Let these elements be y_1, y_2, ..., $y_{2^{384}}$. We introduce probabilistic variables Y_1, Y_2, ..., $Y_{2^{384}}$. The probabilistic value Y_i is 1 if there exists an input x such that $y_i = \text{ChaCha}(x)$ or 0 otherwise. Now, we define a variable $Y = Y_1 + Y_2 + \cdots + Y_{2^{384}}$. The expectation value $E[Y]$ is the average number of elements in the output set.

E[Y] can be evaluated as;

$$
\begin{aligned}
E[Y] &= E[Y_1 + Y_2 + \cdots + Y_{2^{384}}] \\
&= E[Y_1] + E[Y_2] + \cdots + E[Y_{2^{384}}] \\
&= 2^{384} E[Y_1],
\end{aligned}
$$

by the identity of the variables.

On the other hand, $E[Y_1]$ can be calculated as;

$$
\begin{aligned}
E[Y_1] &= 1 - (1 - 2^{-384})^{2^{384}} \\
&\approx 1 - e^{-1} \\
&\approx 0.632.
\end{aligned}
$$

Note that $(1 - 2^{-384})^{2^{384}} \approx e^{-1}$ is the probability that y_1 does not appear in output set with 2^{384} inputs.

Thus, E[Y] can be estimated as $(1 - e^{-1})2^{384}$ and keystream of ChaCha has variety of 2^{383}. We expect to have $2^{191.5}$ keystreams without collision according to the birthday paradox theory. The IETF version of ChaCha can generate up to 2^{32} keystreams within the same key and nonce, and it is expected that no shorter periods would be found.

3.3.8 Side-Channel Attack

We evaluate the security of ChaCha against side channel attacks including single/differential power analysis, cache timing attack, and fault injection attack.

Power Analysis

Single Power Analysis The quarter-round function of ChaCha contains fixed-distance rotations. The fixed-distance rotation operation can be implemented using two bit-shift operations or $(x)_{\lll n} = ((x)_{\ll n}) \oplus ((x)_{\gg (W-n)})$ where W is the bit-length of a word. The bit-shift operations are vulnerable to simple power analysis [Koc09].

Each word of the key is stored into arrays in the initial matrix generation process. An adversary may get the Hamming weight of the words using the power consumption information.

A masking countermeasure can protect these analyses.

Differential Power Analysis We evaluate the security of ChaCha against the correlation power analysis [BCO04] which is a variant of differential power analyses. We assume that an adversary can control the counter $C = (c_0)$ and nonce $N = (n_0, n_1, n_2)$.

The first four instructions in the *quaterround*(x_0, x_4, x_8, x_{12}) in the first round is as followings;

$$\begin{aligned} x_0 &\leftarrow \sigma_0 + k_0 \\ x_{12} &\leftarrow c_0 \oplus (\sigma_0 + k_0) \\ x_{12} &\leftarrow (c_0 \oplus (\sigma_0 + k_0))_{\lll 16} \\ x_8 &\leftarrow k_4 + ((c_0 \oplus (\sigma_0 + k_0))_{\lll 16}) \end{aligned}$$

The adversary focus on the second instruction and measure the power consumption by changing c_0. One can find the value of k_0 using correlation power analysis based on hamming-distance or hamming-weight model. Now, the adversary can control the value of $(c_0 \oplus (\sigma_0 + k_0))_{\lll 16})$ after knowing k_0 and find k_4 from the third instruction. One must use hamming-weight model since the previous value of x_8 is k_4 and unknown.

The first four instructions in the *quaterround*(x_5, x_9, x_{13}, x_1) in the first round is as followings;

$$\begin{aligned} x_5 &\leftarrow k_1 + k_5 \\ x_1 &\leftarrow \sigma_2 \oplus (k_1 + k_5) \\ x_1 &\leftarrow (\sigma_2 \oplus (k_1 + k_5))_{\lll 16} \\ x_{13} &\leftarrow n_0 + ((\sigma_2 \oplus (k_1 + k_5))_{\lll 16}) \end{aligned}$$

The adversary gets $k_1 + k_5$ by controlling n_0 in the fourth instruction.

The first two instructions in the *quaterround*$(x_{10}, x_{14}, x_2, x_6)$ in the first round is as followings;

$$\begin{aligned} x_{10} &\leftarrow k_6 + n_1 \\ x_6 &\leftarrow k_2 \oplus (k_6 + n_1) \end{aligned}$$

The adversary finds k_6 by controlling n_1 in the first instruction, then gets k_2 by controlling $(k_6 + n_1)$.

The first four instructions in the $\mathit{quaterround}(x_{15}, x_{11}, x_{11}, x_7)$ in the first round is as followings;

$$\begin{aligned}
x_{15} &\leftarrow n_2 + \sigma_3 \\
x_{11} &\leftarrow (k_7 \oplus (n_2 + \sigma_3)) \\
x_{11} &\leftarrow (k_7 \oplus (n_2 + \sigma_3))_{\lll 16} \\
x_7 &\leftarrow k_3 + ((k_7 \oplus (n_2 + \sigma_3))_{\lll 16})
\end{aligned}$$

The adversary finds k_7 by controlling $(n_2 + \sigma_3)$ in the second instruction then gets k_2 by controlling $((k_7 \oplus (n_2 + \sigma_3))_{\lll 16})$ in the fourth instruction.

The adversary knows the value of k_0, $k_1 + k_5$, k_2, k_3, k_4, k_6, and k_7 after completing the correlation power analysis. Thus, one can get the entire key by guessing k_1, which impose 2^{32} time complexity. This analysis can be protected with a masking countermeasure.

Cache Timing Attack

ChaCha implementation takes constant time on a huge variety of CPUs. The execution time is input-independent since ChaCha does not contain variant-time operation such as S-box. Cache timing analysis against ChaCha is thus as difficult as pure cryptanalysis of the ChaCha output.

Fault Injection Attack

The key stream of ChaCha is calculated as the addition of the initial matrix X and the matrix $X^{(20)}$ processed by the round function. The initial matrix X or matrix $X^{(20)}$ is output if the addition instruction is skipped by a fault injection attack.

Step 1 Skip one of the `add` instructions that add words of the initial matrix X and the matrix processed by the round-function $X^{(20)}$. The value of one word in the keystream changes if one of the `add` instructions correctly; then, store the word. The step fails if multiple values of words in the keystream change, i.e., multiple instructions or some instructions in round functions are skipped.

Step 2 Repeat Step 1 until obtaining the changed values of all of the word in the keystream.

Step 3 Generate a matrix from the word obtained in Step 1. The matrix is ether X or $X^{(20)}$. We can distinguish the matrix by checking orthogonal words x_0, x_5, x_{10}, and x_{15}. The matrix is X if the orthogonal words are σ_0, σ_1, σ_2, and σ_3 and go to Step 5. The matrix is $X^{(20)}$ otherwise and go to Step 4.

Step 4 Calculate X from $X^{(20)}$ using the inverse round function.

Step 5 Extract the key from the words x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11} in X.

Algorithm 7 Implementation 1 of ChaCha

```
Input: Key $K$, Counter $C$, and Nonce $N$
Output: Keystream $Z$
  $X \leftarrow$ Initial Matrix$(K, C, N)$
  $Z \leftarrow X$
  $X \leftarrow X^{(20)}$
  for $i \leftarrow 0$ to 15 do
    $z_i \leftarrow z_i + x_i$
  end for
  return Z
```

Algorithm 8 Implementation 2 of ChaCha

```
Input: Key $K$, Counter $C$, and Nonce $N$
Output: Keystream $Z$
  $X \leftarrow$ Initial Matrix$(K, C, N)$
  $Z \leftarrow X^{(20)}$
  for $i \leftarrow 0$ to 15 do
    $z_i \leftarrow z_i + x_i$
  end for
  return Z
```

Bar-El et al. [BECN$^+$06] and Trichina and Korkiyan [TK10] demonstrates that instruction skip can be achieved with a laser pulse. Dehbaoui et al. [DDR$^+$12] and Morno et al. [MDH$^+$13] shows instruction skip using an electromagnetic pulse. Korak and Hoefler [KH14], Endo et al. [EHH$^+$14], and Yuce et al [YGS15] proposed instruction skip based on a glitchy clock signal.

Algorithm 7 and 8 shows the implementations of ChaCha. An adversary can obtain the elements of the initial matrix X or the matrix $X^{(20)}$ processed by the round function by skipping the add instruction of the first or second implementation, respectively.

A countermeasure against this sort of attacks is to separate variables; that is, distinct variables store the inputs and output of the addition. Consider an addition $z \leftarrow x+y$. The addition returns the initial value of variable z even if the addition is skipped. Thus, the adversary can get neither the value of variables x and y. Algorithm 9 shows the implementation using countermeasure based on variable separation.

We should note that variable separation in source-code level does not work

Algorithm 9 Countermeasure for fault injection analysis

```
Input: Key $K$, Counter $C$, and Nonce $N$
Output: Keystream $Z$
  $X \leftarrow$ Initial Matrix$(K, C, N)$
  $Y \leftarrow X^{(20)}$
  for $i \leftarrow 0$ to 15 do
    $z_i \leftarrow x_i + y_i$
  end for
  return Z
```

```
int add(int x, int y){
        int z;
        z = x + y;
        return z;
}
```

Figure 3.3: Variable separation in source-code level

```
add:
        pushl   %ebp
        movl    %esp, %ebp
        subl    $16, %esp
        movl    8(%ebp), %edx
        movl    12(%ebp), %eax
        addl    %edx, %eax
        leave
        ret
```

Figure 3.4: Assembly code for x86 architecture

in actual implementation. Figure 3.3 show the source-code of `add` function; and, Figure 3.4 and 3.5 are the assembly code for the x86 and x86-64 architecture, respectively, compiled from the source-code. In the original source-code the inputs are stored in the variables x and y and the output is stored to the variable z; they are separated. On the other hand, the addition `z = x + y` is translated to `addl %edx, %eax` in the assembly code, which means that the addition result of the values in the registers *eax* and *edx* is stored to the register *eax*. The second input is stored into the *eax* in the assembly code in Fig. 3.5 and 3.4; thus, the adversary can get the second input by skipping the `add` instruction. Note that the x86 and x86-64 architecture support only two-operand instruction for addition; thus, we cannot use the variable separation method essentially. However, the x86 and x86-64 architecture have a complicated structure and Ivy

```
add:
        pushq   %rbp
        movq    %rsp, %rbp
        movl    %edi, -20(%rbp)
        movl    %esi, -24(%rbp)
        movl    -20(%rbp), %edx
        movl    -24(%rbp), %eax
        addl    %edx, %eax
        popq    %rbp
        ret
```

Figure 3.5: Assembly code for x86-64 architecture

```
add:
        mov     r2, r0
        add     r0, r1, r2
        bx      lr
```

Figure 3.6: Assembly code for ARM architecture

Bridge microarchitecture has up to 19 stage instruction pipeline. Thus, it is hard to skip a specific instruction in an actual CPUs with the x86 and x86-64 architecture.

The ARM architecture targets resource-constraint devices including IoT. Thus, the ARM has simpler RISC architecture comparing with the x86 and x86-64 architecture. The ARM Cortex-M0, M3, M4 processor have a three-stage instruction pipeline, and the Coretex-M0+ has a two-stage pipeline. Yuce et al. [YGS15] proposed an instruction skip based on a glitchy clock signal against RISC-based CPU with seven stage instruction pipeline. We thus must protect the implementation on the ARM architecture against the fault injection attack.

The ARM architecture three-operand instruction such as `add r0, r1, r2`, which means that the addition result of the values in the registers *r1* and *r2* is stored to the register *r0*. Figure 3.6 shows the assembly code where the countermeasure is used. We need to use assembly or inline-assembly implementation to use the three-operand instruction explicitly.

3.4 Analysis on Poly1305

Bernstein [Ber05b] demonstrated that Poly1305 is ε-almost-Δ-universal where $\varepsilon = 8\lceil L/16\rceil/2^{106}$. We show the definition of ε-almost-Δ-universal and the outline of his proof.

Security Definition Let $(B, +)$ be an Abelian group. A family H of hash functions that maps from a set A to the set B is said to be ε-almost Δ-universal (ε-$A\Delta U$) w.r.t. $(B, +)$, if for any distinct elements $a, a' \in A$ and for all $\delta \in B$:

$$\Pr_{h\in H}[h(a) - h(a') = \delta] \leq \varepsilon.$$

H is Δ-universal (ΔU) if $\varepsilon = 1/|B|$.

Security Proof

Theorem 1. *Poly1305 is ε-$A\Delta U$ where $\varepsilon = 8\lceil L/16\rceil/2^{106}$.*

Proof. Let m, m' be distinct messages, such that $len(m) = len(m') = L$. Define g as a 16-byte string and R be as subset of $\{0, 1, \ldots, 2^{130} - 6\}$. Let U as the set of integers in $[-2^{130} + 6, 2^{130} - 6]$ congruent to g modulo 2^{128}. Note that $\#U \leq 8$.

If $H_r(m) = H_r(m') + g$ then $(m'(r) \mod 2^{130} - 5) - (m(r) \mod 2^{130} - 5) \equiv g(\mod 2^{128})$ so $(\underline{m'}(r) \mod 2^{130} - 5) - (\underline{m}(r) \mod 2^{130} - 5) = u$ for some $u \in U$. Hence r is a root of the polynomial $\underline{m'} - \underline{m} - u$ modulo the prime

$2^{130} - 5$ (by Lemma 1). This polynomial is non-zero by Lemma 2 and has a degree at most $\lceil L/16 \rceil$, so it has at most $\lceil L/16 \rceil$ roots modulo $2^{130} - 5$. Sum over all $u \in U$: there are most $8\lceil L/16 \rceil$ possibilities for r.

Thus, there are at most $8\lceil L/16 \rceil$ integers $r \in R$ such that $H_r(m) = H_r(m')+ g$. Consequently, if $\#R = 2^{106}$, and if r is a uniform random element of R, then $H_r(m) = H_r(m') + g$ with probability at most $8\lceil L/16 \rceil/2^{106}$. □

Lemma 1. $2^{130} - 5$ *is prime.*

Proof. Define $p_1 = (2^{130} - 6)/1517314646$ and $p_2 = (p_1 - 1)/222890620702$. Observe that 37003 and 221101 are prime divisors of $p_2 - 1$; $(37003 \cdot 221101)^2 > p_2$; $2^{p_2 - 1} - 1$ is divisible by p_2; $2^{(p_2-1)/37003} - 1$ and $2^{(p_2-1)/221101} - 1$ are co-prime to p_2; $p_2^2 > p_1$; $2^{p_1 - 1} - 1$ is divisible by p_1; $2^{(p_1-1)/p_2} - 1$ is co-prime to p_1; $p_1^2 > 2^{130} - 5$; $2^{2^{130}-6} - 1$ is divisible by $2^{130} - 5$; and $2^{(2^{130}-6)/p_1} - 1$ is co-prime to $2^{130} - 5$. Hence p_2, p_1, and $2^{130} - 5$ are prime by Pocklington's theorem. □

Lemma 2. *Let m and m' be messages and u be an integer. If the polynomial $\underline{m'} - \underline{m} - u$ is zero modulo $2^{130} - 5$ then $m = m'$.*

Proof. Define $c_1, c_2, \ldots, c_q$ as above, and define $c'_1, c'_2, \ldots c'_q$ for m'.

If $q > q'$ then the coefficient of x_q in $\underline{m'} - \underline{m}$ is $0 - c_1$. By construction c_1 is in $\{1, 2, 3, \ldots, 2^{129}\}$, so it is non-zero modulo $2^{130} - 5$, which is a contradiction. Thus, $q \leq q'$ and $q \geq q'$ similarly; hence, $q = q'$.

If $i \in \{1, 2, \ldots, q\}$ then $c_i - c'_i$ is the coefficient of x^{q+1-i} in $\underline{m'} - \underline{m} - u$, which can be divide by $2^{130} - 5$ using the hypnosis. However, $c_i - c'_i$ is between -2^{129} and 2^{129} by construction; thus, $c_i = c'_i$. In particular, $c_q = c'_q$.

Note that $q = \lceil len(m)/16 \rceil$; thus,L $len(m)$ is between $16q - 15$ and $16q$. The value of $len(m)$ is determined by q and c_q, which is $16q$ if $2^{128} \leq c_q$, $16q - 1$ if $2^{120} \leq c_q < 2^{121}$, $16q - 2$ if $2^{112} \leq c_q < 2^{113}$, ..., $16q - 15$ if $2^8 \leq cq < 2^9$. Hence m' also has $len(m)$ bytes.

Now consider any $j \in \{0, 1, \ldots, len(m)\}$. Write $i = \lceil j/6 \rceil + 1$; then $16i - 16 \leq j- \leq 16i - 1$, and $1 \leq i \leq \lceil len(m)/16 \rceil = q$, so $m[j] = \lfloor c_i/2^{8(j-16i+16)} \rfloor$ mod $256 = \lfloor c'_i/2^{8(j-16i+16)} \rfloor$ mod $256 = m'[j]$. Hence $m = m'$. □

3.5 Analysis on ChaCha20-Poly1305 AEAD

Procter [Pro14] demonstrated that the combination of ChaCha and Poly1305 is a secure authenticated encryption scheme assuming ChaCha is a pseudo-random function (PRF), and Poly1305 is ε-almost-Δ-universal. We show the outline of the security model and proof.

Security Model The accepted definition for a secure authenticated encryption scheme is one that provides both Indistinguishably under Chosen Plaintext Attacks (IND-CPA) and the Integrity of Ciphertexts (INT-CTXT). These notions were introduced by Bellare and Namprempre, and together these properties imply IND-CCA security. In fact, the stronger notion of IND\$-CPA security can be shown to be achieved by this composition.

To model these two notions the adversary is given access to an encryption oracle and a decryption oracle and permitted to make at most q queries to these oracles. The proof proceeds via a series of games and the encryption and decryption oracles in Game i are denoted E^i and D^i respectively. An adversary breaks the IND-CPA security of the scheme if they can distinguish (C, T) generated by E^0 from a random bit-string of the same length (generated by an oracle denoted by the adversary's advantage against the IND$-CPA security of a scheme is measured by

$$\mathbf{Adv}^{\text{ChaCha20-Poly1305 AEAD}}_{\text{IND\$-CPA}} = |\Pr[A^{E^0} \to 1] - \Pr[A^{\$} \to 1]|.$$

An adversary breaks the INT-CTXT security of a scheme if they can forge a ciphertext, i.e. output a tuple (N, A, C, T) with $D_k(N, A, C, T) = (N, A, P) \neq \perp$ where (N, A, C, T) is not the output from an encryption query and D outputs $\perp$ to signify that the input was not a valid ciphertext. The advantage of this adversary is measured by

$$\mathbf{Adv}^{\text{ChaCha20-Poly1305 AEAD}}_{\text{INT-CTXT}} = \Pr[A^{E^0, D^0}\text{forges}].$$

A combined measure of the adversary's advantage against both IND$-CPA and INT-CTXT can be defined as

$$\mathbf{Adv}^{\text{ChaCha20-Poly1305 AEAD}}_{\text{AE}} = |\Pr[A^{E^0, D^0} \to 1] - \Pr[A^{\$, \perp} \to 1]|,$$

where $\perp$ is an oracle that simply returns $\perp$ (representing an invalid ciphertext) an all inputs.

The adversaries that are considered in this section will be restricted to nonce-respecting adversaries. This is a standard restriction for nonce-based authenticated encryption schemes and means that an adversary will never ask encryption queries (N, A, P) and (N, A', P') for $(A, P) \neq (A', P')$. There are no restrictions on the adversary's use of nonces for decryption queries, however without loss of generality, it is assumed that an adversary makes no redundant queries; no query is repeated, and the output from an E query is never inputted to the D oracle or vice versa.

Security Proof

Theorem 2. *ChaCha20-Poly1305 AEAD is IND$-CPA and IND-CTXT secure assuming that ChaCha is a PRF, and Poly1305 is ε-AΔU.*

Proof. It is assumed in this security analysis that no pair (k, N') is ever repeated, where N0 is the 12-byte nonce that is input to the ChaCha block function; this assumption is critical to the security of ChaCha20-Poly1305 AEAD. The draft recognizes that not all protocols will use 12-byte nonces and 'it is up to the protocol document to define how to transform the protocol nonce into a 12-byte nonce; one suggestion is that prepending a constant value could provide a way to expand a shorter nonce to 12 bytes.

If an implementation permits both 12-byte nonces and shorter nonces and an adversary can predict how a short nonce will be expanded to 12 bytes (for example, by guessing the value that will be prepended), then a nonce collision could be forced by querying the encryption oracle using a short N and a 12-byte N' which is the expanded version of N. In what follows, we will assume that all nonces are 12 bytes long and that no (key, nonce) pair is ever repeated to the encryption oracle; the protocol specification, therefore, must prevent nonce collisions of this form.

This section will assume that ChaCha is a PRF with signature ChaCha : $\{0,1\}^{256} \times \{0,1\}^{128} \to \{0,1\}^{512}$, that is, 32-byte keys, 16-byte input, and 64-byte output. This assumption has not been contradicted by any of the existing cryptanalysis of ChaCha and the analysis presented in this note does not concern this assumption.

The proof will also make use of the fact that Poly1305 is an ε-almost-Δ-universal hash function. The proof proceeds via a series of games, specified in Figures 1 and 2. Game 0 defines a combined IND-CPA and INT-CTXT game, with oracles that realize ChaCha20-Poly1305 AEAD. The scheme specified in Game 4 clearly gives no adversary any advantage in either of the IND\$-CPA and INT-CTXT games. The ciphertext and tag are sampled independently of P and uniformly at random from $\{0,1\}^{512}$ (as they would be if generated by \$) and it is impossible for an adversary to query D_4 with anything returning $(N, A, P) \neq \perp$.

The transitions between these games are justified as follows:

Games 0 and 1 If an adversary can distinguish between these two games, then they can distinguish ChaCha from a function chosen uniformly at random from the set of all functions with domain $\{0,1\}^{128}$ and range $\{0,1\}^{512}$. However, we assume that ChaCha is a PRF, so no adversary gains a significant advantage through the transition between these games.

Games 1 and 2 These games are identical, on the condition that the inputs to URF in Game 1 never repeat. The inputs to URF are all of the form $(i||N)$; for each query, N is constant, but i is never reused 3, and no two encryption queries use the same value for N. Therefore the random variables in Games 1 and 2 are identically distributed.

Games 2 and 3 These games are identical unless an adversary submits (N, A, C, T) to their decryption oracle and D_1 returns $(N, A, P) \neq \perp$. However, for each query that an adversary makes, this happens with probability at most ε (because Poly is ε-AΔU). By a standard hybrid argument, the probability that an adversary making at most q queries successfully forge is at most $q\varepsilon$.

Games 3 and 4 The random variables in these games are sampled in different orders; however, the joint distributions are identical and therefore these games are identical.

A standard game-hopping argument allows the probability $\Pr(A^{G(i-1)} \to 1)$ to be bounded in terms of $\Pr(A^{Gi} \to 1)$:

$$\begin{aligned}
&\Pr(A^{G0} \to 1) \le \Pr(A^{G1} \to 1) + \mathbf{Adv}_{\text{PRF}}^{\text{ChaCha}}(B) \\
&\Pr(A^{G1} \to 1) = \Pr(A^{G2} \to 1) \\
&\Pr(A^{G2} \to 1) \le \Pr(A^{G3} \to 1) + q\varepsilon \\
&\Pr(A^{G3} \to 1) = \Pr(A^{G4} \to 1) = \Pr(A^{\$,\perp} \to 1)
\end{aligned}$$

Poly1305 is ε-almost-Δ-universal for $\varepsilon = 8\lceil L/16 \rceil / 2^{106}$, where L denotes the maximum byte length of messages. For this construction L is the largest possible value of $16(\lceil len(A)/16 \rceil + \lceil len(C)/16 \rceil + 1)$, because the specification of ChaCha20-Poly1305 AEAD pads A and C to 16-byte blocks and adds an extra 16 bytes of message denoting the length of additional data and ciphertext. Therefore it can be concluded that for every adversary A there is an adversary B against the PRF security of the ChaCha block function such that:

$$\begin{aligned}\mathbf{Adv}_{\text{AE}}^{\text{Chaha20-Poly1305 AEAD}} &= |\Pr(A^{G0} \to 1) - \Pr(A^{G4} \to 1)| \\ &\leq \mathbf{Adv}_{\text{PRF}}^{\text{ChaCha}}(B) + q\frac{8(\lceil L/16 \rceil)}{2^{106}}.\end{aligned}$$

□

Table 3.1: Comparison of Existing attacks on Salsa20 and ChaCha

Cipher	Round/Key length	Time	Data	Reference
Salsa20	5/256	2^{165}	2^{6}	Crowley [Cro06]
		2^{167}	2^{7}	Velichkov et al. [VMCP12]
		2^{55}	2^{10}	Shi et al. [SZFW12]
		2^{8}	2^{8}	Choudhuri [CM17]
	6/256	2^{177}	2^{16}	Fischer et al. [FMB+06]
		2^{73}	2^{16}	Shi et al. [SZFW12]
		2^{32}	2^{32}	Choudhuri and Maitra [CM17]
	7/128	2^{111}	2^{21}	Aumasson et al. [AFK+08]
		2^{109}	2^{19}	Shi et al. [SZFW12]
	7/256	2^{190}	$2^{11.4}$	Tsunoo et al. [TSK+07]
		2^{151}	2^{26}	Aumasson et al. [AFK+08]
		2^{148}	2^{24}	Shi et al. [SZFW12]
		2^{139}	2^{32}	Choudhuri and Maitra [CM17]
		2^{137}	2^{61}	Choudhuri and Maitra [CM17]
	8/256	2^{255}	$2^{11.4}$	Tsunoo et al. [TSK+07]
		2^{251}	2^{31}	Aumasson et al. [AFK+08]
		2^{250}	2^{27}	Shi et al. [SZFW12]
		$2^{247.2}$	2^{27}	Maitra et al. [MPM15]
		$2^{245.5}$	2^{96}	Maitra [Mai16]
		$2^{244.9}$	2^{96}	Choudhuri and Maitra [CM17]
ChaCha	4/256	2^{6}	2^{6}	Choudhuri and Maitra [CM17]
	4.5/256	2^{12}	2^{12}	Choudhuri and Maitra [CM17]
	5/256	2^{16}	2^{16}	Choudhuri and Maitra [CM17]
	6/128	2^{107}	2^{30}	Aumasson et al. [AFK+08]
		2^{105}	2^{28}	Shi et al. [SZFW12]
	6/256	2^{139}	2^{30}	Aumasson et al. [AFK+08]
		2^{136}	2^{28}	Shi et al. [SZFW12]
		2^{130}	2^{35}	Choudhuri and Maitra [CM17]
		$2^{127.5}$	$2^{37.5}$	Choudhuri and Maitra [CM17]
		2^{116}	2^{116}	Choudhuri and Maitra [CM17]
	7/256	2^{248}	2^{27}	Aumasson et al. [AFK+08]
		$2^{246.5}$	2^{27}	Shi et al. [SZFW12]
		$2^{238.9}$	2^{24}	Maitra [Mai16]
		$2^{233.7}$	2^{96}	Choudhuri and Maitra [CM17]
		2^{233}	2^{28}	Choudhuri and Maitra [CM17]

Table 3.2: Time-Memory-Data Tradeoff Attack

Data (D)	Memory (M)	Preprocessing (P)	Attack (T)	Complexity ($P+T$)
2^{8}	2^{172}	2^{344}	2^{344}	2^{345}
2^{16}	2^{168}	2^{336}	2^{336}	2^{337}
2^{24}	2^{164}	2^{328}	2^{328}	2^{329}
2^{32}	2^{160}	2^{320}	2^{320}	2^{321}
2^{40}	2^{156}	2^{312}	2^{312}	2^{313}
2^{48}	2^{152}	2^{304}	2^{304}	2^{305}
2^{56}	2^{148}	2^{296}	2^{296}	2^{297}
2^{64}	2^{144}	2^{288}	2^{288}	2^{289}
2^{72}	2^{140}	2^{280}	2^{280}	2^{281}
2^{80}	2^{136}	2^{272}	2^{272}	2^{273}
2^{88}	2^{132}	2^{264}	2^{264}	2^{265}
2^{96}	2^{128}	2^{256}	2^{256}	2^{257}
2^{104}	2^{124}	2^{248}	2^{248}	2^{249}
2^{112}	2^{120}	2^{240}	2^{240}	2^{241}
2^{120}	2^{116}	2^{232}	2^{232}	2^{233}
2^{128}	2^{112}	2^{224}	2^{224}	2^{225}
2^{136}	2^{108}	2^{216}	2^{216}	2^{217}
2^{144}	2^{104}	2^{208}	2^{208}	2^{209}
2^{152}	2^{100}	2^{200}	2^{200}	2^{201}
2^{160}	2^{96}	2^{192}	2^{192}	2^{193}

Chapter 4

Conclusion

We evaluated the security of ChaCha20-Poly1305 AEAD. Procter [Pro14] proved that ChaCha20-Poly1305 AEAD is secure if both of ChaCha and Poly1305 are secure as Section 3.4 show the summary of the security proof. Bernstein [Ber05b] demonstrated that Poly1305 is ε-almost-Δ-universal where $= 8\lceil L/16 \rceil / 2^{106}$ as the proof-sketch is given in Section 3.3.

We thus evaluate the security of ChaCha against known cryptanalyses. There are no efficient differential analysis, linear cryptanalysis and distinguishing attack, guess and determine analysis, algebraic attack, and attacks on initialization attacks on ChaCha.

Time-memory-data tradeoff attack theoretically applies to ChaCha. The time complexity to break the original version ChaCha with 64-bit counter and 64-bit IV is 2^{240} using 2^{80} keystreams and 2^{160} memory size. The time complexity is $2^{234.67}$ using 2^{117} keystreams and 2^{176} for the IETF version of ChaCha with 32-bit counter and 96-bit IV. However, there are no time-memory-data tradeoff attack with time complexity less than 2^{256} by limiting the number of keystream to 2^{96}, which is practical assumption.

A naive implementation of ChaCha faces the risk of side-channel analysis including single power analysis, differential power analysis, and fault injection attack. Key recovery based on measurement of power consumption can be deal with a masking countermeasure. We proposed a fault injection attack on software implementation by skipping the addition (`add`) instructions to make the keystream matrix Z from the initial matrix X and the matrix $X^{(20)}$ processed by the round function. The attack can also be protected with variable separation countermeasure.

Therefore, we conclude that we cannot find any weaknesses in ChaCha20-Poly1305 AEAD.

Table 4.1: Summary of Security Analysis

Algorithm	Attack	Evaluation
ChaCha	Differential Analysis • Rotational Cryptanalysis • Boomerang Attack	No attack found
	Linear Cryptanalysis	No attack found
	Distinguishing Attack	No attack found
	Guess and Determine Analysis	No attack found
	Time-Memory-Data Tradeoff Attack	Protected Practically
	Algebraic Attack	No attack found
	Attacks on Initialization Process	No attack found
	Single Power Analysis	Protected Practically
	Difference Power Analysis	Protected Practically
	Cache Timing Attack	No attack found
	Fault Injection Analysis	Protected Practically
Poly1305	N/A	Provable Secure
ChaCha20-Poly1305	N/A	Provable Secure

Bibliography

[AFK$^+$08] Jean-Philippe Aumasson, Simon Fischer, Shahram Khazaei, Willi Meier, and Christian Rechberger. New Features of Latin Dances: Analysis of Salsa, ChaCha, and Rumba. In *Prof. of 15th Fast Software Encryption Workshop (FSE 2008), Lecture Notes in Computer Science*, volume 5086, pages 470–488, 2008. https://eprint.iacr.org/2007/472.

[Bab95] Steve H. Babbage. Improved exhaustive search attacks on stream ciphers. In *Prof. of European Convention on Security and Detection, IEE Conference*, volume 408, pages 161–166, 1995.

[BC04] Eli Biham and Rafi Chen. Near-Collisions of SHA-0. In *Proc. of Advances in Cryptology (CRYPTO 2004), Lecture Notes in Computer Science*, volume 3152, pages 290–305, 2004.

[BC14] Eli Biham and Yaniv Carmeli. An Improvement of Linear Cryptanalysis with Addition Operations with Applications to FEAL-8X. In *Proc. of 21st International Conference on Selected Areas in Cryptography, Lecture Notes in Computer Science*, volume 8781, pages 59–76, 2014.

[BCO04] E. Brier, C. Clavier, and F Olivier. Correlation Power Analysis with a Leakage Model. In *Proc. of Workshop on Cryptographic Hardware and Embedded Systems (CHES 2004)*, pages 135–152, 2004.

[BECN$^+$06] Hagai Bar-El, Hamid Choukri, David Naccache, Michael Tunstall, and Claire Whelan. The Sorcerer's Apprentice Guide to Fault Attacks. *Proceeding of the IEEE*, 94(2), 2006.

[Ber05a] Daniel Julius Bernstein. Salsa20 security. https://cr.yp.to/snuffle/security.pdf, 2005.

[Ber05b] Daniel Julius Bernstein. The Poly1305-AES message-authentication code. In *Prof. of 12th Fast Software Encryption Workshop (FSE 2005), Lecture Notes in Computer Science*, volume 3557, pages 32–49, 2005. https://cr.yp.to/papers.html#poly1305.

[Ber08a] Daniel Julius Bernstein. ChaCha, a variant of Salsa20. In *SASC 2008, The State of the Art of Stream Ciphers, Workshop Record, ECRYPT Network of Excellence in Cryptology*, 2008. https://cr.yp.to/papers.html#chacha.

[Ber08b] Daniel Julius Bernstein. The Salsa20 family of stream ciphers. In *New Stream Cipher Designs, The eSTREAM Finalists, Lecture Notes in Computer Science*, volume 4986, pages 84–97, 2008. https://cr.yp.to/papers.html#salsafamily.

[BS93] Eli Biham and Adi Shamir. *Differential Cryptanalysis of the Data Encryption Standard.* Springer-Verlag, 1993.

[BS00] Alex Biryukov and Adi Shamir. Cryptanalytic Time/Memory/Data Tradeoffs for Stream Ciphers. In *Prof. of Advances in Cryptology (Asiacrypt 2000)*, volume 1976, pages 1–13, 2000.

[BS05] An Braeken and Igor Semaev. The ANF of the Composition of Addition and Multiplication mod 2^n with a Boolean Function. In *Proc. of 12th Fast Software Encryption Workshop (FSE 2005), Lecture Notes in Computer Science*, volume 3557, pages 112–125, 2005.

[CM16] Arka Rai Choudhuri and Subhamoy Maitra. Differential Cryptanalysis of Salsa and ChaCha — An Evaluation with a Hybrid Model. https://eprint.iacr.org/2016/377, 2016.

[CM17] Arka Rai Choudhuri and Subhamoy Maitra. Significantly Improved Multi-bit Differentials for Reduced Round Salsa and ChaCha. *IACR Transactions on Symmetric Cryptology*, 2017(2):261–287, 2017.

[Cro06] Paul Crowley. Truncated differential cryptanalysis of five rounds of Salsa20. In *SASC 2006, Stream Ciphers Revisited, Workshop Record, ECRYPT Network of Excellence in Cryptology*, 2006. https://eprint.iacr.org/2005/375.

[DDR$^+$12] A. Dehbaoui, J. M. Dutertre, B. Robisson, P. Orsatelli, P. Maurine, and A. Tria1. Injection of transient faults using electromagnetic pulses Practical results on a cryptographic system. In *Proc. of Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2012)*, pages 7–15, 2012.

[EHH$^+$14] Sho Endo, Naofumi Homma, Yuichi Hayashi, Junko Takahashi, Hitoshi Fuji, and Takafumi Aoki. Constructive Side-Channel Analysis and Secure Design. In *Proc. of Constructive Side-Channel Analysis and Secure Design (COSADE 2014), Lecture Notes in Computer Science*, volume 8622, pages 214–228, 2014.

[FMB$^+$06] Simon Fischer, Willi Meier, Come Berbain, Jean-Fran¥ccois Biasse, and Matthew J. B. Robshaw. Non-randomness in eSTREAM candidates Salsa20 and TSC-4. In *Proc. of 7th International Conference on Cryptology in India (INDOCRYPT 2006), Lecture Notes in Computer Science*, volume 4329, pages 2–16, 2006.

[Gol97] Jovan Dj. Golitć. Cryptanalysis of Alleged A5 Stream Cipher. In *Prof. of Eurocrypt 1997, Lecture Notes in Computer Science*, volume 1233, pages 239–255, 1997.

[Hel80] Martin E. Hellmann. A Cryptanalytic Time-Memory Trade-Off. *IEEE Transaction on Information Theory*, 26(4), 1980.

[HS05] Jin Hong and Palash Sarkar. Rediscovery of Time Memory Trade-offs. https://eprint.iacr.org/2005/090, 2005.

[KH14] Thomas Korak and Michael Hoefler. On the Effects of Clock and Power Supply Tampering on Two Microcontroller Platforms. In *Proc. of Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2014)*, pages 8–17, 2014.

[KI16] Imamura Kazuya and Tetsu Iwata. Nonce-Misuse and Decryption-Misuse Resistance of ChCha20-Poly1305. 3D1-2, 2016 Symposium on Cryptography and Information Security (in Japanese), 2016.

[KN10] Dmitry Khovratovich and Ivica Nikolić. Rotational Cryptanalysis of ARX. In *Proc. of 17th Fast Software Encryption Workshop (FSE 2010), Lecture Notes in Computer Science*, volume 6147, pages 333–346, 2010.

[KNP$^+$15] Dmitry Khovratovich, Ivica Nikolić, Josef Pieprzyk, Przemysław Sokołowski, and Ron Steinfeld. Rotational Cryptanalysis of ARX Revisited. In *Proc. of 22nd Fast Software Encryption Workshop (FSE 2015), Lecture Notes in Computer Science*, volume 9054, pages 519–536, 2015.

[Knu95] Lars R Knudsen. Truncated and higher order differentials. In *Proc. of Workshop on Fast Software Encryption (FSE 1994), Lecture Notes in Computer Science*, volume 1008, pages 196–211, 1995.

[Koc09] Cetin Kaya Koc, editor. *Cryptographic Engineering.* Springer, 2009.

[Leu16] Gaëtan Leurent. Improved Differential-Linear Cryptanalysis of 7-round Chaskey with Partitioning. In *Proc. of Advances in Cryptology (EUROCRYPT 2016), Lecture Notes in Computer Science*, volume 9665, pages 344–371, 2016.

[LH94] Susan K. Langford and Martin E. Hellman. Differential-linear cryptanalysis. In *Proc. of Advances in Cryptology (CRYPTO 1994), Lecture Notes in Computer Science*, volume 839, pages 17–25, 1994.

[LM01] Helger Lipmaa and Shiho Moriai. Efficient Algorithms for Computing Differential Properties of Addition. In *Proc. of 8th Fast Software Encryption Workshop (FSE 2001), Lecture Notes in Computer Science*, pages 336–350. Springer, 2001.

[Mai16] Subhamoy Maitra. Chosen IV cryptanalysis on reduced round ChaCha and Salsa. *Discrete Applied Mathematics*, 208:88–97, 2016. https://eprint.iacr.org/2015/698.

[MDH$^+$13] Nicolas Moro, Amine Dehbaoui, Karine Heydemann, Bruno Robisson, and Emmanuelle Encrenaz. Electromagnetic fault injection:

towards a fault model on a 32-bit microcontroller. In *Proc. of Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2013)*, pages 77–88, 2013.

[MP13] Nicky Mouha Mouha and Bart Preneel. Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20. https://eprint.iacr.org/2013/328, 2013.

[MPM15] Subhamoy Maitra, Goutam Paul, and Willi Meier. Salsa20 Cryptanalysis: New Moves and Revisiting Old Styles. https://eprint.iacr.org/2015/217, 2015.

[NL15] Y. Nir and A. Langley. ChaCha20 and Poly1305 for IETF Protocols. RFC 7539 (Informational), may 2015.

[Pro14] Gordon Procter. A Security Analysis of the Composition of ChaCha20 and Poly1305. https://eprint.iacr.org/2014/613, 2014.

[Sar09] Palash Sarkar. On Approximating Addition by Exclusive OR. https://eprint.iacr.org/2009/047, 2009.

[SZFW12] Zhenqing Shi, Bin Zhang, Dengguo Feng, and Wenling Wu. Improved Key Recovery Attacks on Reduced-Round Salsa20 and ChaCha. In *Proc. of Information Security and Cryptology (ICISC 2012), Lecture Notes in Computer Science*, volume 7839, pages 337–351, 2012.

[TK10] Elena Trichina and Roman Korkikyan. Multi Fault Laser Attacks on Protected CRT-RSA. In *Proc. of Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2010)*, pages 75–86, 2010.

[TSK$^+$07] Yukiyasu Tsunoo, Teruo Saito, Hiroyasu Kubo, Tomoyasu Suzaki, and Hiroki Nakashima. Differential cryptanalysis of Salsa20/8. In *SASC 2007, The State of the Art of Stream Ciphers, Workshop Record, ECRYPT Network of Excellence in Cryptology*, 2007.

[VMCP12] Vesselin Velichkov, Nicky Mouha, Christophe De Canni¥‘ere, and Bart Preneel. UNAF: a special set of additive differences with application to the differential analysis of ARX. In *Prof. of 19th Fast Software Encryption Workshop (FSE 2012), Lecture Notes in Computer Science*, volume 7549, pages 287–305, 2012.

[Wag99] David Wagner. The Boomerang Attack. In *Proc. of 6th Fast Software Encryption Workshop (FSE 1999), Lecture Notes in Computer Science*, volume 1636, pages 156–170, 1999.

[YGS15] Bilgiday Yuce Yuce, Nahid Farhady Ghalaty, and Patrick Schaumont. Improving Fault Attacks on Embedded Software Using RISC Pipeline Characterization. In *Proc. of Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC 2015)*, pages 97–108, 2015.