International Journal of Bifurcation and Chaos

Design and smartphone implementation of H.264-codec video chaotic duplex communications

Baoju Chen*, Simin Yu†, Zeqing Zhang‡
School of Automation, Guangdong University of Technology,
Guangzhou 510006, P. R. China
**bogychan@foxmail.com*
†*siminyu@163.com*
‡*2111704029@mail2.gdut.edu.cn*

David Day-Uei Li
Faculty of Science, University of Strathclyde,
Glasgow G4 0RE, U. K.
David.Li@strath.ac.uk

Jinhu Lü§
School of Automation Science and Electrical Engineering, Beihang University,
Beijing 100191, P. R. China
jhlu@iss.ac.cn

Received (to be inserted by publisher)

In this paper, an H.264-codec-based video chaotic secure duplex communication scheme was designed and its smartphone implementation was also carried out. First, an improved self-synchronous chaotic stream cipher algorithm equipped with a sinusoidal modulation, a multiplication, a modulo operation and a round down operation (SCSCA-SMMR) was developed. Using the sinusoidal modulation and multiplication, the improved algorithm can resist the divide-and-conquer attack by traversing multiple non-zero component initial conditions (DCA-TMNCIC). Meanwhile, also by means of the round down operation and modulo operation, on the premise that the DCA-TMNCIC does not work, the original keys cannot be further deciphered only by the known-plaintext attack, the chosen-plaintext attack and the chosen-ciphertext attack, respectively. Then, the Android low-level multimedia support infrastructure MediaCodec class was used to access low-level media encoder/decoder components and the H.264 hardware encoding/decoding was performed on real-time video, so the video chaotic encryption and decryption can be realized in real-time by smartphones. Security analysis and smartphone experiment results verify the effectiveness of the proposed method.

Keywords: video chaotic secure duplex communication; divide-and-conquer attack; chosen-ciphertext attack; H.264 hardware encoding/decoding; smartphone implementation.

§Author for correspondence

1. Introduction

In 1989, Robert Matthews proposed a chaotic encryption algorithm for the first time by studying the Logistic map [Matthews, 1989]. Since then, chaotic encryption algorithms and their applications in multimedia secure communication have attracted wide attention. Chaotic cryptosystems are mainly classified into two types: open-loop chaotic cryptosystems [Arslan & Junaid, 2018; Hua & Zhou, 2016; Norouzi *et al.*, 2014; Parvin *et al.*, 2016; Song & Qiao, 2015; Xu & Tian, 2019; Yang *et al.*, 2015] and closed-loop feedback chaotic cryptosystems [Elhosany *et al.*, 2012; Niu *et al.*, 2017; Shahzadi *et al.*, 2019; Wu & Chua, 1993; Zhang *et al.*, 2019; Zhu *et al.*, 2018]. For the open-loop chaotic cryptosystem, since the encrypted signals are not fed back into the encryption process or the chaotic system, the plaintext and the chaotic sequence remain entirely independent. Therefore, the cryptanalyst can obtain the equivalent keys according to cryptographic analysis methods such as the known-plaintext attack, the chosen-plaintext attack, and the chosen-ciphertext attack [Li *et al.*, 2019; Özkaynak, 2018; Wen & Yu, 2019; Wen *et al.*, 2019; Zhang & Yu, 2019]. For example, in [Arslan & Junaid, 2018], an image encryption cryptosystem based on binary bit planes extraction and multiple chaotic maps (IEC-BPMC) was proposed, and a classical structure named "permutation-diffusion" was adopted. However, the encryption sequences used for bit-level permutation and bit-wise XOR diffusion are not related to plain images, which were mentioned in [Wen & Yu, 2019]. Moreover, once the diffusion part is deciphered, IEC-BPMC will degenerate into a permutation-only process. But according to the analysis results in [Jolfaei *et al.*, 2015; Li & Lo, 2011; Zhang *et al.*, 2018], it was proved to be insecure. Thus, the equivalent diffusion keys and the equivalent permutation keys can be obtained by the chosen-plaintext attack method and the divide-and-conquer strategy, respectively. In [Xu & Tian, 2019], an image chaotic encryption algorithm based on orthogonal Latin cubes and bit cubes was given, in which a structure named "permutation-diffusion-permutation" was employed. But [Zhang & Yu, 2019] pointed out that the generation of Latin cubes in this algorithm is independent of the plain image, and the corresponding number of bits in the cipher image changes regularly in the diffusion stage when any bit in the plain image changes. Because of this fundamental flaw, the equivalent keys can be acquired by the chosen plaintext attack and the differential attack. Besides, there are also some chaotic encryption schemes, in which the secret keys are related to the plaintext [Jain & Rajpal, 2016; Niyat *et al.*, 2017; Ye & Huang, 2015]. However, these schemes are open-loop systems essentially, and the equivalent keys can also be recovered by the chosen-plaintext attack [Dou *et al.*, 2017; Li *et al.*, 2018a,b]. For example, in [Li *et al.*, 2018b], the security analysis of an encryption algorithm based on hybrid hyper-chaos and cellular automata proposed in [Niyat *et al.*, 2017] was given, and it pointed out that the generation of key stream is related to the sum of plaintext pixels, while keeping the sum of pixel values of each color channel unchanged, the equivalent permutation keys can be obtained by adjusting individual pixel values of the chosen plaintexts, and all possibilities of the equivalent diffusion keys can be obtained by using the chosen-plaintext attack.

Compared with the open-loop chaotic cryptosystem, the closed-loop feedback chaotic cryptosystem is related to the plaintexts or the ciphertexts, increasing the difficulty of deciphering encryption algorithms. Specifically, in the closed-loop cryptosystem, the equivalent keys cannot be obtained directly by the cryptanalyst like the open-loop cryptosystem. The cryptanalyst can only obtain the original keys by employing cryptanalysis, which is undoubtedly more difficult than obtaining the equivalent keys. Note that there is an important class of self-synchronous chaotic stream cipher algorithms in closed-loop feedback chaotic cryptosystems [Chen *et al.*, 2018a; Gan *et al.*, 2017; Lin *et al.*, 2015], and the main features are as follows: (1) The ciphertexts containing the plaintext information are fed back into the chaotic system to realize self-synchronization, which are independent of the initial conditions of senders and receivers; (2) In the actual channels (such as LAN and WAN), synchronization can be restored automatically when external interferences occur. In fact, existing communication systems always run in actual channels, so the self-synchronous ability is necessary to meet actual needs; (3) In multimedia chaotic secure communication systems realized by FPGA, ARM, SOPC, and other hardware platforms, the self-synchronous chaotic stream cipher reveals practical application values, especially when the chaotic encrypted signal needs to realize long-distance real-time transmission through actual channels; (4) Since the round down operation and modulo operation are used to obtain the lower 8 bits of the state variables to encrypt the plaintext, the cryptanalyst can only obtain the lower 8 bits of the state variables through cryptanalysis, but not all the

information. Therefore, when the initial values are not used as the attack conditions, the original keys are difficult to decipher directly by using the chosen-ciphertext attack only. However, the main problem of this kind of self-synchronous chaotic stream cipher algorithms is that the original chaotic iterative equation will degenerate into an asymptotically stable iterative one when the ciphertext is selected as a fixed value, so the explicit function of state variables and secret keys can be obtained. Based on this idea, and according to the chosen-ciphertext attack, the secret keys can be deciphered by the single key decipher algorithm [Lin *et al.*, 2018a]. In particular, the cryptanalyst can traverse single non-zero component initial conditions to decipher the secret keys of the encryption algorithm by the divide-and-conquer attack if the initial conditions are not used as the secret keys [Lin *et al.*, 2018b]. On the one hand, for further improving the security performance of self-synchronous chaotic stream cipher algorithms, the SCSCA-SMMR was proposed in this paper. On the other hand, to make a more comprehensive security analysis for self-synchronous chaotic stream cipher algorithms, an improved divide-and-conquer attack method with stronger attack strength was also proposed, which is based on the divide-and-conquer attack by traversing single non-zero component initial conditions (DCA-TSNCIC) given in [Lin *et al.*, 2018b]. The main feature of the improved divide-and-conquer attack method is that all possible choices for multiple non-zero component initial conditions are traversed through the exhaustive method. With the new arrangements mentioned above, the improved self-synchronous chaotic stream cipher algorithm can resist the DCA-TMNCIC and also can solve the security problems of the self-synchronous chaotic stream cipher algorithms proposed in [Chen *et al.*, 2018a; Gan *et al.*, 2017; Lin *et al.*, 2015]. The security analysis comparisons between the self-synchronous chaotic stream cipher algorithms proposed in recent years and the SCSCA-SMMR proposed in this paper are given below, as shown in Table 1.

From Table 1, with the SCSCA-SMMR, a chaotic encryption and decryption scheme based on H.264 bitstream was designed and its smartphone video chaotic secure communication was also realized. Video capturing, video previewing, H.264 hardware encoding, chaotic encrypting, and network sending were realized at smartphone senders. Network receiving, chaotic decrypting, H.264 hardware decoding, and video displaying were implemented at smartphone receivers. The main features are as follows:

(1) An improved algorithm 3-D SCSCA-SMMR has been developed suitable for mobile phone implementations. Chaotic encryption, closed-loop feedback and transmission of ciphertexts can be performed by taking the lower 8 bits derived from the product of a state variable and a sinusoidally modulated state variable. The cryptanalysis results indicated that with the sinusoidal modulation and multiplication, the improved algorithm can resist the DCA-TMNCIC. In addition, the round down operation and modulo operation have been adopted in the improved algorithm, when the DCA-TMNCIC does not work, the original keys cannot be further deciphered only by the known-plaintext attack, the chosen-plaintext attack and the chosen-ciphertext attack, respectively.

(2) An encryption-decryption scheme of 3-D SCSCA-SMMR based on H.264 bitstream has been proposed to encrypt only the network abstract layer unit (NALU) payloads of H.264 bitstream, which can improve the encryption efficiency and ensure the integrity of H.264 format.

(3) The Android low-level multimedia support infrastructure MediaCodec class has been used to access low-level media coder/decoder components, and H.264 hardware encoding/decoding can be performed on real-time video. Furthermore, the original H.264 bitstream was encrypted at smartphone senders, the encrypted H.264 bitstream was decrypted at smartphone receivers, and the real-time protocol (RTP) and user datagram protocol (UDP) were used to transmit the encrypted H.264 bitstream through the duplex network.

The rest of the paper is organized as follows: Section 2 introduces the 3-D SCSCA-SMMR and gives its corresponding security analysis. Section 3 gives the design method of chaotic encryption and decryption based on H.264 bitstream. Section 4 discusses the H.264-codec-based smartphone implementation scheme for video chaotic secure duplex communication in detail. Section 5 demonstrates the hardware experiments and gives the discussions of this work. Finally, Section 6 concludes the paper.

Table 1. Comparisons of security analysis of self-synchronous chaotic stream cipher algorithms

Encryption algorithms	The main features of the algorithms	Analysis methods	Analysis results
8-D self-synchronous chaotic stream cipher algorithm [Lin *et al.*, 2015].	The RGB tricolors of the video pixels were encrypted by the lower 8 bits derived from the state variables $x_1(k), x_2(k), x_3(k)$ with the round down operation and modulo operation.	Combination of the known-plaintext attack, the chosen-ciphertext attack, and the DCA-TSNCIC [Lin *et al.*, 2018b].	Exception of the secret keys multiplied by ciphertext and related to nonlinear functions were deciphered.
3-D self-synchronous chaotic stream cipher algorithm [Gan *et al.*, 2017].	The speech data was encrypted by the lower 8 bits derived from the state variable $x_1(k)$ with the round down operation and modulo operation.	Combination of the chosen-ciphertext attack and the single key decipher algorithm [Lin *et al.*, 2018a].	Exception of the secret keys multiplied by ciphertext and related to nonlinear functions were deciphered.
3-D self-synchronous chaotic stream cipher algorithm [Chen *et al.*, 2018a].	The RGB tricolors of the video pixels were encrypted by the lower 8 bits derived from the product of two state variables $x_1(k), x_2(k)$ with the round down operation and modulo operation.	Combination of the chosen-ciphertext attack and the DCA-TMNCIC.	Exception of the secret keys multiplied by ciphertext and related to nonlinear functions were deciphered.
This work	The H.264 bitstream was encrypted by the lower 8 bits derived from the product of a state variable $x_1(k)$ and a sinusoidally modulated state variable $\sin(x_2(k))$ with the round down operation and modulo operation.	Combination of the known-plaintext attack, the chosen-plaintext attack, the chosen-ciphertext attack and the DCA-TMNCIC.	The secret keys cannot be deciphered temporarily, and a more powerful attack method should be required.

2. The 3-D SCSCA-SMMR and its security analysis

According to Table 1, an 8-D self-synchronous chaotic stream cipher algorithm was proposed in [Lin *et al.*, 2015], where the ciphertexts were fed back into the chaotic system to realize self-synchronization. The main feature is that the RGB tricolors of the video pixels were encrypted by the lower 8 bits derived from the state variables $x_1(k), x_2(k), x_3(k)$ with the round down operation and modulo operation, so the cryptanalyst can only obtain the lower 8 bits of the corresponding state variables. However, the algorithm cannot resist the DCA-TSNCIC, which using the set of eight single initial conditions choices $\{(c_1, 0, \cdots, 0), (0, c_2, 0, \cdots, 0), \cdots, (0, \cdots, 0, c_7, 0), (0, 0, \cdots, 0, c_8)\}$ and setting $c_i\,(i = 1, 2, \cdots, 8)$ as $2^7, 2^{7+8}, 2^{7+16}, 2^{7+24}, 2^{7+32}, 2^{7+40}, 2^{7+48}, 2^{7+56}$ to decipher the original keys [Lin *et al.*, 2018b]. For this problem, the algorithm proposed in [Gan *et al.*, 2017] was further designed in [Chen *et al.*, 2018a] by using the lower 8 bits derived from the product of two state variables for encryption, and the corresponding encryption sequence was derived as $E(k) = \left(\mathrm{mod}\left(\left\lfloor \frac{x_1^{(e)}(k) \times x_2^{(e)}(k)}{2^{27}} \right\rfloor, 2^8\right)\right)$. But the cryptanalysis results show that although the algorithm can resist the DCA-TSNCIC, it cannot resist the DCA-TMNCIC. To cope with this problem, in this section, an improved algorithm 3-D SCSCA-SMMR was developed, and the cryptanalysis proved that the improved algorithm can resist the DCA-TMNCIC.

2.1. *The description of 3-D SCSCA-SMMR*

According to [Chen *et al.*, 2018a] and [Gan *et al.*, 2017], the iterative equation of a chaotic encryption system was derived as

$$\begin{cases} x_1^{(e)}(k+1) = a_{11}x_1^{(e)}(k) + a_{12}x_2^{(e)}(k) + a_{13}x_3^{(e)}(k) \\ x_2^{(e)}(k+1) = a_{21}p(k) + a_{22}x_2^{(e)}(k) + a_{23}x_3^{(e)}(k) \\ x_3^{(e)}(k+1) = a_{31}p(k) + a_{32}x_2^{(e)}(k) + a_{33}x_3^{(e)}(k) + \varepsilon\sin(\sigma \times p(k)) \end{cases}. \tag{1}$$

Similarly, the iterative equation of a chaotic decryption system was derived as

$$\begin{cases} x_1^{(d)}(k+1) = a_{11}x_1^{(d)}(k) + a_{12}x_2^{(d)}(k) + a_{13}x_3^{(d)}(k) \\ x_2^{(d)}(k+1) = a_{21}p(k) + a_{22}x_2^{(d)}(k) + a_{23}x_3^{(d)}(k) \\ x_3^{(d)}(k+1) = a_{31}p(k) + a_{32}x_2^{(d)}(k) + a_{33}x_3^{(d)}(k) + \varepsilon\sin(\sigma \times p(k)) \end{cases}. \tag{2}$$

In Eqs. (1) - (2), $k = 0, 1, 2, 3, 4\cdots$, $a_{11} = 0.205$, $a_{12} = -0.595$, $a_{13} = 0.265$, $a_{22} = -0.125$, $a_{23} = 0.595$, $a_{31} = 0.33$, $a_{32} = -0.33$, $a_{33} = 0.47$, $\varepsilon = 3\times10^{12}$, and $\sigma = 2\times10^5$ denote the secret keys, $p(k)$ denotes the ciphertext, $x_i^{(e)}(k)\,(i = 1, 2, 3)$ denote the state variables of the chaotic encryption system, $x_i^{(d)}(k)\,(i = 1, 2, 3)$ denote the state variables of the chaotic decryption system.

According to Eqs. (1) - (2), the 3-D SCSCA-SMMR can be described as follows:

(1) Chaotic sequence for encryption

In the improved algorithm, the chaotic sequence $x_1^{(e)}(k)$ and $x_2^{(e)}(k)$ used for encryption can be generated by Eq. (1). By using the sinusoidal modulation, multiplication, round down operation, and modulo operation, the encryption sequence $E(k)$ can be derived as

$$E(k) = \left(\mathrm{mod}\left(\left\lfloor x_1^{(e)}(k) \times \sin(x_2^{(e)}(k))\right\rfloor, 2^8\right)\right), \tag{3}$$

where $k = 0, 1, 2, \cdots$, $\mathrm{mod}\left(\lfloor\cdot\rfloor, 2^8\right)$ denotes module operation, and $\lfloor\cdot\rfloor$ denotes round down operation.

(2) Encryption process

By encrypting the plaintext $m(k)$ with $E(k)$, the ciphertext $p(k)$ can be derived as

$$\begin{aligned} p(k) &= E(k) \oplus m(k) \\ &= \left(\mathrm{mod}\left(\left\lfloor x_1^{(e)}(k) \times \sin(x_2^{(e)}(k))\right\rfloor, 2^8\right)\right) \oplus m(k)' \end{aligned} \tag{4}$$

where $k = 0, 1, 2, \cdots$, $\oplus$ denotes XOR operation.

(3) Chaotic sequence for decryption

In the improved algorithm, the chaotic sequence $x_1^{(d)}(k)$ and $x_2^{(d)}(k)$ used for decryption can be generated by Eq. (2). By using the sinusoidal modulation, multiplication, round down operation, and modulo operation, the decryption sequence $D(k)$ can be derived as

$$D(k) = \left(\mathrm{mod}\left(\left\lfloor x_1^{(d)}(k) \times \sin(x_2^{(d)}(k))\right\rfloor, 2^8\right)\right), \tag{5}$$

where $k = 0, 1, 2, \cdots$.

(4) Decryption process

By decrypting the ciphertext $p(k)$ with $D(k)$, the decrypted data $\hat{m}(k)$ can be derived as

$$\begin{aligned} \hat{m}(k) &= D(k) \oplus p(k) \\ &= \left(\mathrm{mod}\left(\left\lfloor x_1^{(d)}(k) \times \sin(x_2^{(d)}(k))\right\rfloor, 2^8\right)\right) \oplus p(k)' \end{aligned} \tag{6}$$

where $k = 0, 1, 2, \cdots$.

2.2. *Security analysis*

Since the sinusoidal modulation and multiplication are involved in the improved algorithm given by Eqs. (1) - (6), the 3-D SCSCA-SMMR can resist the DCA-TMNCIC. Meanwhile, also the round down operation and modulo operation are included in the improved algorithm, when the DCA-TMNCIC does not work, the original keys cannot be further deciphered only by the known-plaintext attack, the chosen-plaintext attack the chosen-ciphertext attack, respectively. The cryptanalysis in detail is as follows:

(1) According to the exhaustive method, all possible choices for multiple non-zero component initial conditions can be traversed. By choosing the initial conditions $x_i(0) = c_i\,(i = 1, 2, 3)$, the set of all initial condition choices can be derived as

$$(x_1(0), x_2(0), x_3(0)) = \{(c_1, 0, , 0), (0, c_2, 0), (0, 0, c_3), (c_1, c_2, 0), (c_1, 0, c_3), (0, c_2, c_3), (c_1, c_2, c_3)\}. \quad (7)$$

From Eq. (7), it is noted that the set of all possible choices for single initial conditions choices $\{(c_1, 0, 0), (0, c_2, 0), (0, 0, c_3)\}$ is the subset of the set of all possible choices for multiple non-zero component initial conditions. Therefore, in terms of attack strength, the DCA-TMNCIC is stronger than the DCA-TSNCIC proposed in [Lin *et al.*, 2018b].

(2) According to the known-plaintext attack method, the cryptanalyst should obtain not only $m(k)$, but also the corresponding $p(k)$. With the first known plaintext $m(0)$, the corresponding first known ciphertext $p(0)$ can be derived as

$$\begin{aligned} p(0) &= E(0) \oplus m(0) \\ &= \left(\mathrm{mod}\left(\left\lfloor x_1^{(e)}(0) \times \sin(x_2^{(e)}(0))\right\rfloor, 2^8\right)\right) \oplus m(0)' \end{aligned} \quad (8)$$

where $E(0)$ denotes the first encryption sequence value.

By choosing $x_i(0) = c_i\,(i = 1, 2, 3)$ and substituting $p(0)$ into Eq. (1) or Eq. (2), the first iterative result can be derived as

$$\begin{cases} x_1(1) = a_{11}c_1 + a_{12}c_2 + a_{13}c_3 \\ x_2(1) = a_{21}p(0) + a_{22}c_2 + a_{23}c_3 \\ x_3(1) = a_{31}p(0) + a_{32}c_2 + a_{33}c_3 + \varepsilon\sin(\sigma \times p(0)) \end{cases}. \quad (9)$$

By substituting $x_1(1)$ and $x_2(1)$ in Eq. (9) into Eq. (4), the second known ciphertext $p(1)$ can be derived as

$$\begin{aligned} p(1) &= E(1) \oplus m(1) \\ &= \left(\mathrm{mod}\left(\left\lfloor x_1^{(e)}(1) \times \sin(x_2^{(e)}(1))\right\rfloor, 2^8\right)\right) \oplus m(1) \\ &= \left(\mathrm{mod}\left(\left\lfloor (a_{11}c_1 + a_{12}c_2 + a_{13}c_3) \times \sin(a_{21}p(0) + a_{22}c_2 + a_{23}c_3)\right\rfloor, 2^8\right)\right) \oplus m(1) \end{aligned}, \quad (10)$$

where $E(1)$ denotes the second encryption sequence value, $m(1)$ denotes the second known plaintext.

According to Eq. (10), the explicit relation among the secret keys $a_{ij}(i = 1, 2; j = 1, 2, 3)$, $m(1)$, and $p(1)$ can be derived as

$$\left(\mathrm{mod}\left(\left\lfloor (a_{11}c_1 + a_{12}c_2 + a_{13}c_3) \times \sin(a_{21}p(0) + a_{22}c_2 + a_{23}c_3)\right\rfloor, 2^8\right)\right) = p(1) \oplus m(1). \quad (11)$$

According to the DCA-TMNCIC method, by substituting Eq. (7) into Eq. (11), then setting $c_i = c\,(i = 1, 2, 3)$, the corresponding explicit relationships among $a_{ij}\,(i = 1, 2, 3; j = 1, 2, 3)$, $m(1)$, and $p_i(1)\,(i = 1, 2, \cdots, 7)$ can be derived as

$$\begin{cases} \left(\mathrm{mod}\left(\lfloor a_{11}c \times \sin(a_{21}p(0))\rfloor, 2^8\right)\right) = p_1(1) \oplus m(1) \\ \left(\mathrm{mod}\left(\lfloor a_{12}c \times \sin(a_{21}p(0) + a_{22}c)\rfloor, 2^8\right)\right) = p_2(1) \oplus m(1) \\ \left(\mathrm{mod}\left(\lfloor a_{13}c \times \sin(a_{21}p(0) + a_{23}c)\rfloor, 2^8\right)\right) = p_3(1) \oplus m(1) \\ \left(\mathrm{mod}\left(\left\lfloor (a_{11}c + a_{12}c) \times \sin((a_{21}p(0) + a_{22}c))\right\rfloor, 2^8\right)\right) = p_4(1) \oplus m(1) \\ \left(\mathrm{mod}\left(\left\lfloor (a_{11}c + a_{13}c) \times \sin((a_{21}p(0) + a_{23}c))\right\rfloor, 2^8\right)\right) = p_5(1) \oplus m(1) \\ \left(\mathrm{mod}\left(\left\lfloor (a_{12}c + a_{13}c) \times \sin(a_{22}p(0) + a_{23}c)\right\rfloor, 2^8\right)\right) = p_6(1) \oplus m(1) \\ \left(\mathrm{mod}\left(\left\lfloor (a_{11}c + a_{12}c + a_{13}c) \times \sin(a_{21}p(0) + a_{22}c + a_{23}c)\right\rfloor, 2^8\right)\right) = p_7(1) \oplus m(1) \end{cases}. \quad (12)$$

According to Eq. (12), when the 3-D SCSCA-SMMR is attacked by using the DCA-TMNCIC, we cannot obtain the direct relationships among a_{ij} $(i=1,2,3; j=1,2,3)$, $m(k)$, and $p(k)$. The sinusoidal modulation, multiplication, round down operation and modulo operation are involved in the improved algorithm, so the relationships between a_{ij} $(i=1,2,3; j=1,2,3)$ become more complicated. Specifically, with the product of $x_1(k)$ and $\sin(x_2(k))$, the common factor c of each formula in Eq. (12) cannot be extracted, so we cannot set the suitable initial conditions to obtain the correct estimated values of a_{ij} $(i=1,2,3; j=1,2,3)$, such as setting c as $2^7, 2^{7+8}, 2^{7+16}, 2^{7+24}, 2^{7+32}, 2^{7+40}, 2^{7+48}, 2^{7+56}$ to decipher the secret keys proposed in [Lin *et al.*, 2018b]. In addition, only the lower 8 bits derived from the product of $x_1(k)$ and $\sin(x_2(k))$ can be obtained, so we cannot further decipher the original keys only by the known-plaintext attack when the DCA-TMNCIC does not work.

With the cryptanalysis mentioned above, we can know that the secret keys cannot be deciphered by employing the combination of the known-plaintext attack and the DCA-TMNCIC. Similarly, the same conclusion can be obtained by utilizing the combination of the chosen-plaintext attack and the DCA-TMNCIC.

(3) According to the chosen-ciphertext attack method, the cryptanalyst can choose the ciphertext in the favor of deciphering, and also the cryptanalyst can obtain the corresponding plaintext. In order to eliminate the influence introduced by the nonlinear functions in Eq. (1) and Eq. (2), we select $p(k)=0$ and the original chaotic equation can be derived as

$$\begin{cases} x_1(k+1) = a_{11}x_1(k) + a_{12}x_2(k) + a_{13}x_3(k) \\ x_2(k+1) = a_{22}x_2(k) + a_{23}x_3(k) \\ x_3(k+1) = a_{32}x_2(k) + a_{33}x_3(k) \end{cases}, \tag{13}$$

where $k=0,1,2,\cdots$. It is obvious that with the chosen-ciphertext attack, Eq. (13) is degenerate as a linear system. According to Eq. (6), the decryption operation can be derived as

$$\begin{aligned} \hat{m}(k) &= \left(\mathrm{mod}(\lfloor x_1(k)\times\sin(x_2(k))\rfloor, 2^8)\right) \oplus p(k) \\ &= \left(\mathrm{mod}(\lfloor x_1(k)\times\sin(x_2(k))\rfloor, 2^8)\right) \oplus 0 \\ &= \left(\mathrm{mod}(\lfloor x_1(k)\times\sin(x_2(k))\rfloor, 2^8)\right) \end{aligned}. \tag{14}$$

By substituting $x_i(0)=c_i$ $(i=1,2,3)$ into Eq. (13), the first iterative result can be derived as

$$\begin{cases} x_1(1) = a_{11}c_1 + a_{12}c_2 + a_{13}c_3 \\ x_2(1) = a_{22}c_2 + a_{23}c_3 \\ x_3(1) = a_{32}c_2 + a_{33}c_3 \end{cases}. \tag{15}$$

By substituting $x_1(1)$ and $x_2(1)$ in Eq. (15) into Eq. (14), the second decrypted data $\hat{m}(1)$ can be derived as

$$\begin{aligned} \hat{m}(1) &= \left(\mathrm{mod}(\lfloor x_1(1)\times\sin(x_2(1))\rfloor, 2^8)\right) \oplus p(1) \\ &= \left(\mathrm{mod}(\lfloor x_1(1)\times\sin(x_2(1))\rfloor, 2^8)\right) \oplus 0 \\ &= \left(\mathrm{mod}(\lfloor (a_{11}c_1 + a_{12}c_2 + a_{13}c_3)\times\sin((a_{22}c_2 + a_{23}c_3))\rfloor, 2^8)\right) \end{aligned}. \tag{16}$$

According to the DCA-TMNCIC method, by substituting Eq. (7) into Eq. (16), then setting $c_i = c$ $(i=1,2,3)$, the corresponding explicit relationships among a_{ij} $(i=1,2,3; j=1,2,3)$ and $\hat{m}_i(1)$ $(i=1,2,\cdots,7)$ can be derived as

$$\begin{cases} \hat{m}_1(1) = \left(\mathrm{mod}\left(\lfloor x_1(1)\times\sin(x_2(1))\rfloor, 2^8\right)\right) = 0 \\ \hat{m}_2(1) = \left(\mathrm{mod}\left(\lfloor x_1(1)\times\sin(x_2(1))\rfloor, 2^8\right)\right) = \left(\mathrm{mod}\left(\lfloor a_{12}c\times\sin(a_{22}c)\rfloor, 2^8\right)\right) \\ \hat{m}_3(1) = \left(\mathrm{mod}\left(\lfloor x_1(1)\times\sin(x_2(1))\rfloor, 2^8\right)\right) = \left(\mathrm{mod}\left(\lfloor a_{13}c\times\sin(a_{23}c)\rfloor, 2^8\right)\right) \\ \hat{m}_4(1) = \left(\mathrm{mod}\left(\lfloor x_1(1)\times\sin(x_2(1))\rfloor, 2^8\right)\right) = \left(\mathrm{mod}\left(\lfloor (a_{11}c + a_{12}c)\times\sin(a_{22}c)\rfloor, 2^8\right)\right) \\ \hat{m}_5(1) = \left(\mathrm{mod}\left(\lfloor x_1(1)\times\sin(x_2(1))\rfloor, 2^8\right)\right) = \left(\mathrm{mod}\left(\lfloor (a_{11}c + a_{13}c)\times\sin(a_{23}c)\rfloor, 2^8\right)\right) \\ \hat{m}_6(1) = \left(\mathrm{mod}\left(\lfloor x_1(1)\times\sin(x_2(1))\rfloor, 2^8\right)\right) = \left(\mathrm{mod}\left(\lfloor (a_{12}c + a_{13}c)\times\sin((a_{22}c + a_{23}c))\rfloor, 2^8\right)\right) \\ \hat{m}_7(1) = \left(\mathrm{mod}\left(\lfloor x_1(1)\times\sin(x_2(1))\rfloor, 2^8\right)\right) = \left(\mathrm{mod}\left(\lfloor (a_{11}c + a_{12}c + a_{13}c)\times\sin((a_{22}c + a_{23}c))\rfloor, 2^8\right)\right) \end{cases}. \tag{17}$$

According to Eq. (17), when the 3-D SCSCA-SMMR is attacked by using the DCA-TMNCIC, we cannot obtain the direct relationships among a_{ij} $(i = 1, 2, 3; j = 1, 2, 3)$, $\hat{m}(k)$. The sinusoidal modulation, multiplication, round down operation and modulo operation are involved in the improved algorithm, so the relationships between a_{ij} $(i = 1, 2, 3; j = 1, 2, 3)$ become more complicated. Specifically, with the product $x_1(k)$ and $\sin(x_2(k))$, the common factor c of each formula in Eq. (17) cannot be extracted, so we cannot set the suitable initial conditions to obtain the correct estimated values of a_{ij} $(i = 1, 2, 3; j = 1, 2, 3)$, such as setting c as $2^7, 2^{7+8}, 2^{7+16}, 2^{7+24}, 2^{7+32}, 2^{7+40}, 2^{7+48}, 2^{7+56}$ to decipher the secret keys proposed in [Lin *et al.*, 2018b]. In addition, only the lower 8 bits derived from the product of $x_1(k)$ and $\sin(x_2(k))$ can be obtained, so we cannot further decipher the original keys only by the chosen-ciphertext attack when the DCA-TMNCIC does not work.

With the cryptanalysis mentioned above, we can know that the secret keys cannot be deciphered by employing the combination of the chosen-ciphertext attack and the DCA-TMNCIC.

3. The design method of chaotic encryption-decryption based on H.264 bitstream

In this section, a design method of chaotic encryption-decryption based on H.264 bitstream was adopted to realize the 3-D SCSCA-SMMR proposed in Section 2, as shown in Fig. 1.

From Fig.1, we can know that the data transmission between the sender and the receiver is transmitted

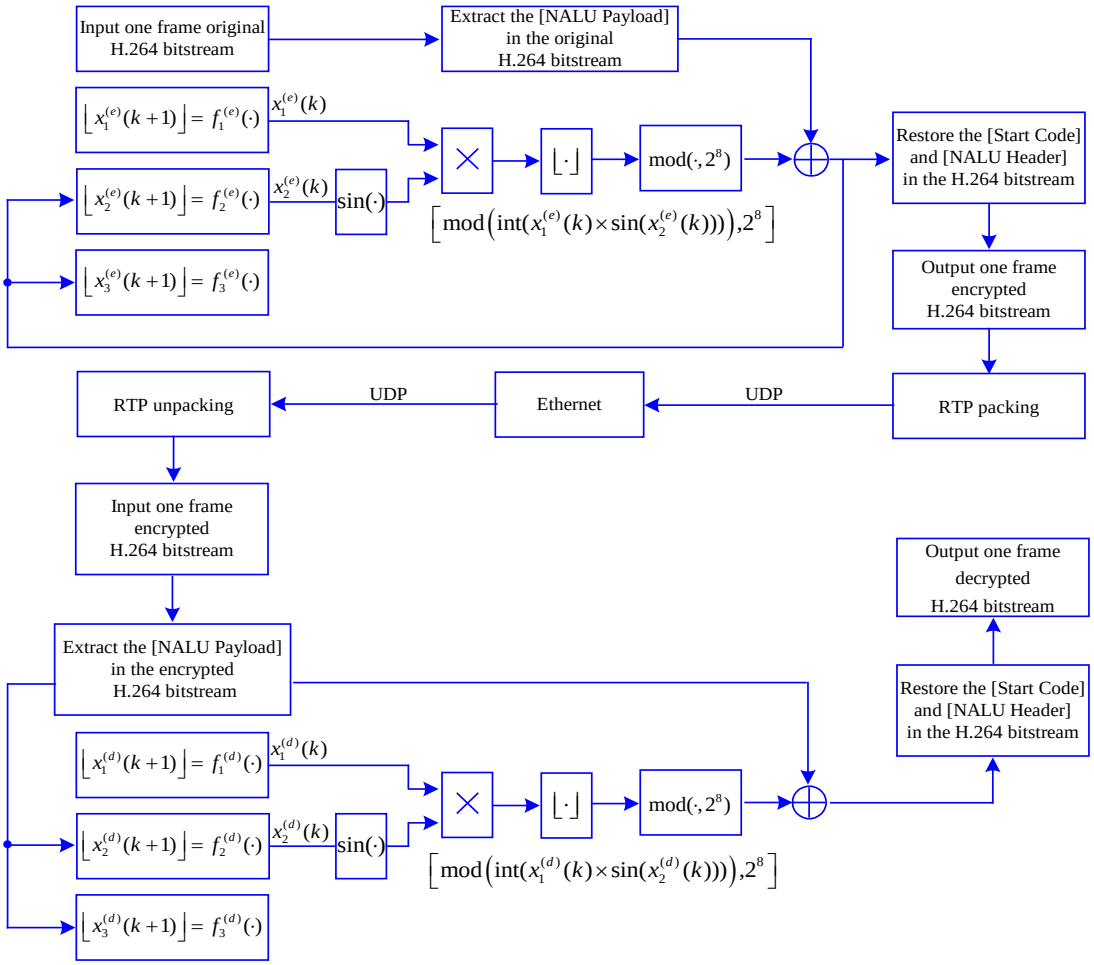

Fig. 1. Design method of chaotic stream cipher encryption-decryption based on H.264 bitstream

through the actual channels (such as LAN and WAN), and it usually suffers from signal interferences and data loss in actual transmission processes. However, the chaotic precise synchronization of self-synchronous chaotic stream cipher algorithms requires a transient process. In this process, if some of H.264 bitstream are lost in actual network channel transmissions, the decrypted H.264 bitstream will be inconsistent with the encrypted H.264 bitstream, which may damage the H.264 format and fail to decode the H.264 bitstream at the receiver. Note that an original H.264 NALU consists of [Start Code], [NALU Header] and [NALU Payload], where [Start Code] and [NALU Header] are codec recognition formats of the H.264 bitstream, [NALU Payload] is the payload of the H.264 bitstream. Therefore, in this scheme, for ensuring the integrity of H.264 format and improving the encryption efficiency, only [NALU Payload] of the H.264 bitstream is extracted for encryption, and [Start Code] and [NALU Header] are not encrypted. The design principles are as follows:

(1) At the sender, first, one frame H.264 bitstream is received from the encoder. Then, only [NALU Payload] of the original H.264 bitstream is extracted for encryption. When the chaotic encryption is completed, [Start Code] and [Header] are restored, and one frame encrypted H.264 bitstream is generated. Finally, the RTP is used for the packet, and the UDP is used for network transmission.

(2) At the receiver, first, one frame encrypted H.264 bitstream packetized by the RTP is received from the network through the UDP. Then, only [NALU Payload] of the encrypted H.264 bitstream is extracted for decryption. When the chaotic decryption is completed, [Start Code] and [NALU Header] are restored, and one frame decrypted H.264 bitstream is generated. Finally, the decrypted H.264 bitstream is transmitted to the decoder for decoding.

4. The H.264-codec-based smartphone implementation scheme for video chaotic secure duplex communication

4.1. *Overall design scheme*

In this section, an overall design scheme of the H.264-codec-based smartphone implementation scheme for video chaotic secure duplex communication was proposed. As shown in Fig. 2, the whole system includes two smartphone senders, two smartphone receivers, and a network transmission. Video capturing, video previewing, H.264 hardware encoding, chaotic encrypting, and network sending are realized at smartphone senders. Network receiving, chaotic decrypting, H.264 hardware decoding, and video displaying are implemented at smartphone receivers. The duplex communication of network is realized by the combination of RTP and UDP. The design principles are as follows:

(1) There are four video data buffer queues in the whole system, one for sending video data and one for receiving video data in each of two smartphones. Video capturing, video previewing, H.264 hardware encoding, chaotic encrypting, network sending, network receiving, chaotic decrypting, H.264 hardware decoding, and video displaying are performed in different threads. Asynchronous communication of data is realized through queue cache, and parallel processing of data is implemented in a multithread architecture.

(2) There are five modules both at the Sender-1 and Sender-2, including a video capture module, a video preview module, an H.264 hardware encoding module, a chaotic encryption module, and an RTP packing and UDP sending module. In the video capture module, the original video data is captured, then the original video is previewed and pushed to the data buffer queue in the video preview module. In the H.264 hardware encoding module, the Android low-level multimedia support infrastructure MediaCodec class is adopted to access low-level media coder/decoder components, and the original video data is pulled from the data buffer queue to perform the H.264 hardware encoding, then the original H.264 bitstream is obtained and pushed to the data buffer queue. In the chaotic encryption module, the original H.264 bitstream pulled from the data buffer queue is encrypted, then the encrypted H.264 bitstream is pushed to the data buffer queue. In the RTP packing and UDP sending module, the encrypted H.264 bitstream is pulled from the data buffer queue for RTP packetization and UDP network sending. Note that Sender-1 transmits the encrypted H.264 bitstream to Receiver-1, Sender-2 transmits the encrypted H.264 bitstream to Receiver-2.

(3) There are four modules both at Receiver-1 and Receiver-2, including a UDP receiving and RTP packing module, a chaotic decryption module, an H.264 hardware decoding module, and a video display

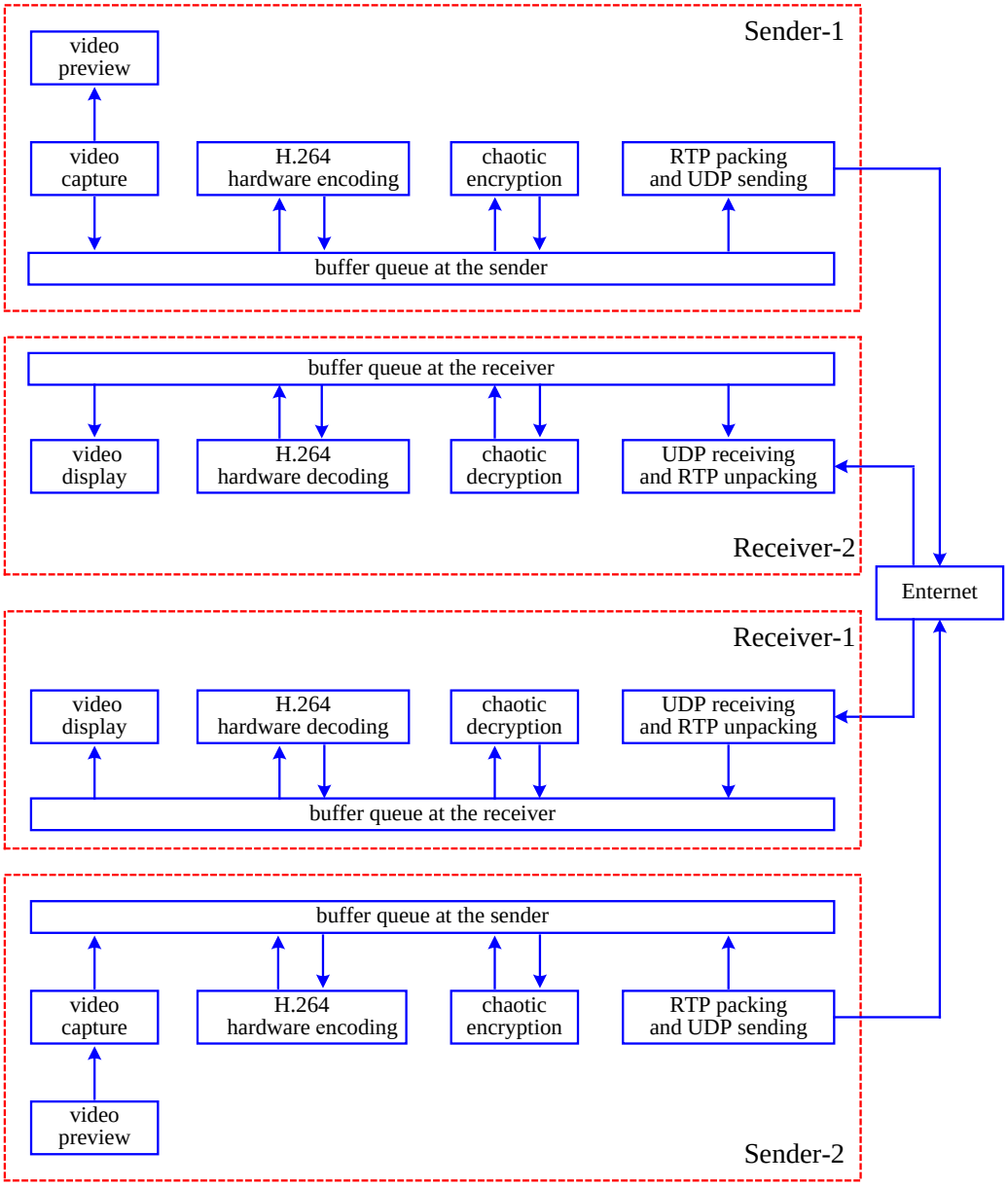

Fig. 2. Diagram of smartphone implementation of an H.264-codec-based video chaotic secure duplex communication system

module. In the UDP receiving and RTP unpacking module, the RTP packets received from Sender-1 and Sender-2 are unpacked and restored to the encrypted H.264 bitstream, then the encrypted H.264 bitstream is pushed to the data buffer queue. In the chaotic decryption module, the H.264 bitstream pulled from the data buffer queue is decrypted, then the decrypted H.264 bitstream is pushed to the data buffer queue. In the H.264 hardware decoding module, the Android low-level multimedia support infrastructure MediaCodec class is adopted to access low-level media coder/decoder components, and the decrypted H.264 bitstream is pulled from the data buffer queue to perform the H.264 hardware decoding, then the decoded video data is obtained and pushed to the data buffer queue. In the video display module, the decoded video data pulled from the data buffer queue is displayed on the surface.

4.2. *Design and implementation of sender program architecture*

According to Fig. 2, the diagram of the sender program architecture is given, as shown in Fig. 3, including six classes that work on different threads: Main class, Camera class, Encoder class, Sender class, and BufferQueue class. Note that each class contains a multi-level program architecture, colors red, green, yellow, purple, and black in order from program architecture level 1 to 5. In each class, the execution priority of program architecture is level 1 > level 2 > level 3 > level 4 > level 5. In addition, the arrows represent the direction of data interaction between different program architecture and classes, and the functions are realized by the methods indicated by the arrows.The functions of each class and method are detailed below:

The Main class realizes the threat creation and controls the start and the end of each thread, including:

(1) intCameraSuface(): Initialize the parameters of the camera.

(2) setVideoFrameListener(): Set a listener for the camera to capture the original video data.

(3) startCamera(): Turn on the camera.

(4) createEncoder(): Create an encoder thread and a new buffer queue to cache the data in the H.264 hardware encoding module via new EncoderBufferQueue().

(5) createEncrypter(): Create a chaotic encryption thread and a new buffer queue to cache the data in the chaotic encryption module via new EncrypterBufferQueue().

(6) createSender(): Create a network sending thread and a new buffer queue to cache the data in the RTP packing and UDP sending module via new SenderBufferQueue().

(7) stop(): Stop all threads.

The Camera class realizes the video capture and preview, including:

(1) setPreviewParams(): Set the parameters for the original video preview.

(2) setPreviewCallback(CaptureYuvStr): Capture the original video data cyclically.

(3) onCaptureRawFrameCallback(byte[] buffer): Callback the original video data.

(4) pushEncoderQueue(): Push the original video data to the buffer queue.

(5) startPreview(): Strat the video preview.

The Encoder class adopts the Android low-level multimedia support infrastructure MediaCodec class to access low-level media coder/decoder components, and performs the H.264 hardware encoding on the real-time video, including:

(1) new Encoder(EncoderBufferQueue): Create an encoder object.

(2) configEncoder(): Set the parameters of the encoder.

(3) setEncoderListener(): Set an encoding listener.

(4) stratEncoderThread(): Strat the encoding thread.

(5) pullEncoderQueue(): Pull the original video data from the buffer queue.

(6) mediaCodecEncoding(): Encode the original video data cyclically, and then obtain the original H.264 bitstream.

(7) callbackEncryptdata(): Callback the original H.264 bitstream.

(8) pushEncrypterQueue(): Push the original H.264 bitstream in the buffer queue.

The Encrypter class realizes the chaotic encryption of the original H.264 bitstream, including:

(1) new Encrypter(EncrypterBufferQueue): Create an encrypter object.

(2) stratEncrypterThread(): Strat the chaotic encryption thread.

(3) pullEncrypterQueue(): Pull the original H.264 bitstream from the buffer queue.

(4) encrryptBitStr(): Extract and encrypt [NALU payload] of the original H.264 bitstream.

(5) callbackSenddata(): Callback the encrypted H.264 bitstream.

(6) pushSenderQueue(): Push the encrypted H.264 bitstream to the buffer queue.

The Sender class realizes the RTP packing and UDP sending, including:

(1) new Sender(SenderBufferQueue): Create a network sending object.

(2) startSenderThread(): Strat the network sending thread.

(3) startRtpPacketizer(): Pack the encrypted H.264 bitstream by RTP.

(4) udpSend(): Send the RTP packets by UDP.

The BufferQueue class realizes the buffer queues to cache the data for each thread.

4.3. *Design and implementation of receiver program architecture*

According to Fig. 2, the diagram of receiver program architecture is given, as shown in Fig. 4, including four classes that work on different threads: Main class, Receiver class, Decrypter class, Decoder class, and BufferQueue class. Note that each class contains a multi-level program architecture, colors red, green, yellow, and purple in order from program architecture level 1 to 4. In each class, the execution priority of program architecture is level 1 > level 2 > level 3 > level 4. In addition, the arrows represent the direction of data interaction between different program architecture and classes, and the functions are realized by the methods indicated by the arrows.The functions of each class and method are detailed belows:

The Main class realizes the threat creation and controls the start and the end of each thread, including:

(1) creatReceiver(): Create a network receiving thread.

(2) creatDecrypter(): Create a decrypter thread and a new buffer queue to cache the data in the chaotic decryption module via new DecrypterBufferQueue().

(3) creatDecoder(): Create a decoder thread and a new buffer queue to cache the data in the H.264 hardware decoding module via new DecoderBufferQueue().

(4) stop(): Stop all threads.

The Receiver class realizes the UDP receiving and RTP unpacking of the encrypted H.264 bitstream packets, including:

(1) new Receiver (ReceiverBufferQueue): Create a network receiving object.

(2) startReceiverThread(): Strat the network receiving thread.

(3) startUdpReceiver(): Unpack the RTP packets received from the network, and then restore them to the encrypted H.264 bitstream.

(4) startFrameConstruct(): Reconstruct the unpacked encrypted H.264 bitstream to a completed frame H.264 bitstream.

(5) callbackFramedata(): Callback each frame of reconstructed encrypted H.264 bitstream.

(6) pushDecrypterQueue(): Push each frame of reconstructed encrypted H.264 bitstream to the buffer queue.

The Decrypter class realizes the chaotic decryption of the encrypted H.264 bitstream, including:

(1) new Decrypter(DecrypterBufferQueue): Create a decrypter object.

(2) stratDecrypterThread(): Strat the chaotic decryption thread.

(3) pullDecrypterQueue(): Pull the encrypted H.264 bitstream from the buffer queue.

(4) decryptBitStr(): Extract and decrypt [NALU payload] of the encrypted H.264 bitstream.

(5) callbackDecryptdata(): Callback the decrypted H.264 bitstream.

(6) pushDecrypterQueue(): Push the decrypted H.264 bitstream to the buffer queue.

The Decoder class adopts the Android low-level multimedia support infrastructure MediaCodec class to access low-level media coder/decoder components, and performs the H.264 hardware decoding on the decrypted H.264 bitstream, including:

(1) new Decoder(DecoderBufferQueue): Create a decoder object.

(2) setDecoderListener(): Set a decoding listener.

(3) startDecoderThread(): Strat the decoding thread.

(4) pullDecrypterQueue(): Pull the decrypted H.264 bitstream from the buffer queue.

(5) findAvcNalSpsAndPPS(): Find the sequence parameter set (SPS) and picture parameter set (PPS) in each frame of decrypted H.264 bitstream.

(6) configDecoder(): Set the parameters of the decoder.

(7) mediaCodecDecoding(): Decode the decrypted H.264 bitstream cyclically, and then obtain the decoded video data.

(8) startRenderThread(): Display the decoded video by using open graphics library for embedded systems (OpenGL ES).

The BufferQueue class realizes buffer queues to cache the data for each thread.

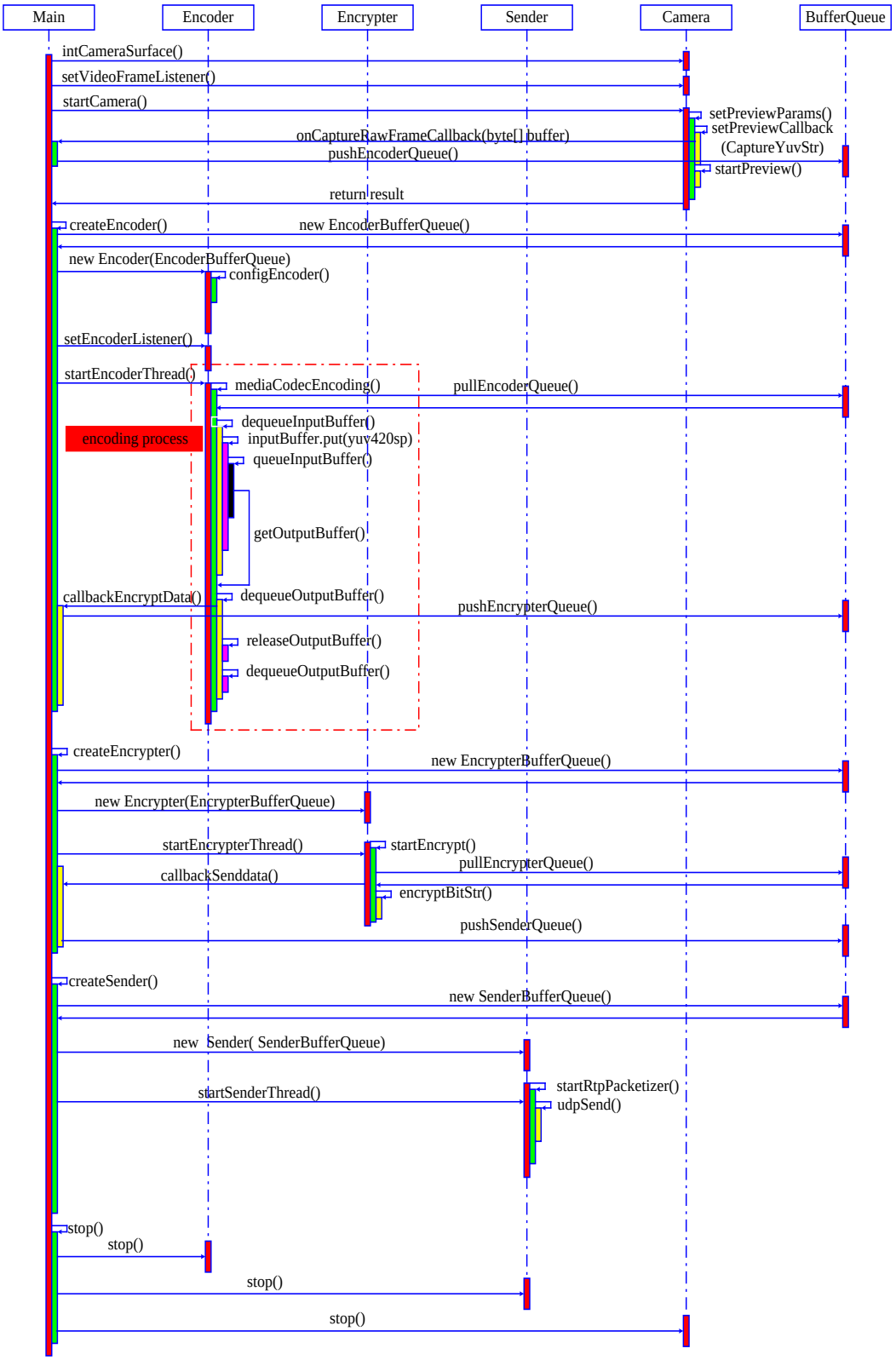

Fig. 3. Diagram of sender program architecture

Fig. 4. Diagram of receiver program architecture

5. Hardware implementation results and discussions

According to the design scheme given in Section 3 and Section 4, two smartphones named Huawei Mate10 Pro with HiSilicon Kirin 970, Octa-core CPU (4 × Cortex A73 2.36 GHz + 4 × Cortex A53 1.80 GHz) + i7 co-processor, Mali-G72 MP12 GPU and 6 GB RAM + 128 GB ROM memory were selected as communication devices. In the experiment, the IP addresses were set as 192.168.1.101 and 192.168.1.111, at the sender and at the receiver respectively. Note that each smartphone can be used as both a sender and a receiver, under the duplex communication mode. When the secret keys both at the sender and at the receiver are matched, the encrypted H.264 bitstream can be accurately decrypted at the receiver, as shown in Fig. 5. The original videos are displayed on the interfaces of Sender-1 and Sender-2. The encrypted H.264 bitstreams received from Sender-1 and Sender-2 are accurately decrypted, decoded and displayed on the interfaces of Receiver-1 and Receiver-2, respectively. But even if only one secret key is mismatched, while other secret keys are exactly matched, the encrypted H.264 bitstream cannot be decrypted at the receiver, as shown in Fig. 6. The encrypted H.264 bitstreams received from Sender-1 and Sender-2 are unsuccessfully decrypted, decoded and displayed on the interfaces of Receiver-1 and Receiver-2, respectively. When the network communication status is normal, and the encrypted H.264 bitstream is accurately decrypted at the receiver.

As we know, transmission frame rate is an important performance of video chaotic secure communication system. In the experiment, we used the number of frames calculation method under a fixed time to test the transmission frame rate at different resolutions. The formula for calculating the transmission frame rate can be derived as

$$f_{ps} \approx \frac{frameNum}{fixedTime} \approx \frac{frameNum}{curTime - lastTime}(\text{f/s}), \tag{18}$$

where f_{ps} denoted the number of frames transmitted per second, $frameNum$ denoted the number of frames transmitted in fixed time, $fixedTime$ denoted the fixed running time, $lastTime$ denoted the initial system time before the frame count, $curTime$ denoted the current system time. According to Eq. (18), one gets the corresponding calculation algorithm, given by Algorithm 1. When calling the function System.currentTimeMills() in the algorithm, the current system time will be returned.

Algorithm 1 The algorithm for calculating transmission frame rate

Input: $fixedTime$
Output: f_{ps}
$lastTime \leftarrow$ System.currentTimeMills()
while TURE **do**
  $frameCount{+}{+}$
  $curTime \leftarrow$ System.currentTimeMills()
  if $(curTime - lastTime > fixedTime)$ **then**
    $frameNum \leftarrow frameCount$
    $f_{ps} \leftarrow frameNum/(curTime - lastTime)$
    return f_{ps}
  end if
end while

Moreover, the efficiency of the scheme realized on smartphones is mainly for consumed time of H.264 encoding/decoding, chaotic encryption/decryption and network transmission. In the experiment, we used the time calculation method under a fixed number of frames to test the consumed time of each module at different resolutions. The algorithm for calculating the consumed time of 100 frames of video is given by Algorithm 2.

According to the Algorithm 1 and Algorithm 2, the transmission frame rate and the consumed time of each fountion module is tested at different resolutions, as shown in Table 2. From Table 2, with the increase of resolution, the consumed time of chaotic encryption/decryption also increases, but little effect

Algorithm 2 The algorithm for calculating the consumed time of 100 frames of video

```
Output: consumedTime
lastTime ← System.currentTimeMills()
while TURE do
  runing function module
  frameCount++
  if (frameCount == 100) then
    curTime ← System.currentTimeMills()
    consumedTime ← curTime − lastTime
    return consumedTime
  end if
end while
```

on the consumed time of H.264 encoding/decoding and network transmission. It can be seen that the computational complexity of encryption algorithm is one of the important parts affecting the real-time performance.

With the discussion of the proposed scheme mentioned above, we can know that comparing with numerical simulations reported in some studies, hardware implementations for video chaotic secure communication are much more difficult to realize because of several real-time requirements and a large amount of data encryption and transmission. At present, there are many studies in video chaotic secure communication systems realized by ARM, SOPC, FPGA, and other hardware platforms[Lin *et al.*, 2015; Chen *et al.*, 2018a,b], but few reports for video chaotic secure communication on mobile devices. Compared with the previous hardware implementation schemes, the advantages of the scheme proposed in this paper are as follows:(1) It is the first time to realize video chaotic secure wireless communication system in duplex mode, in which both the sender and the receiver adopt the smartphone of Android system. The proposed scheme can serve as a good application example of chaotic secure communications for smartphone and other mobile devices in the future.(2) Regardless of the performance differences between different hard platforms, firstly, for greatly compressing data, the H.264 hardware encoding is performed on real-time video. Secondly, for ensuring the integrity of H.264 format and improving the encryption efficiency of the H.264 bitstream, only the payload is encrypted. Thirdly, for reducing the network transmission delay, the strategy of combining RTP and UDP is adopted. Finally, for improving the work efficiency of the system, the queue cache structure and multithread architecture are employed. All of these methods can be well optimized the real-time performance of the video chaotic communication system.

Furthermore, since the sinusoidal modulation and multiplication are adopted in the 3D SCSCA-SMMR to meet high security, the computation complexity of the algorithm is also increased. Considering the computation complexity of 3D SCSCA-SMMR and the performance of hardware devices restrict the further improvement of real-time performance, as shown in Table 2. Only the resolution is 320×240, the transmission frame rate after encryption/decryption is higher than 25 f/s. With higher resolutions such as 640×480, the transmission frame rate after encryption/decryption is lower than 25 f/s. Thus, our future work is to study a better strategy to balance the real-time performance and the computation complexity and security of encryption algorithm, we can propose an encryption algorithm with high security and low computation complexity to meet the high real-time requirements at higher resolutions.

Table 2. Test results of transmission frame rate and consumed time of each module at different resolutions

Resolution	H.264 encoding	chaotic encrypting	RTP packing and UDP sending	H.264 decoding	chaotic decrypting	UDP receiving and RTP unpacking	Transmission frame rate
320×240	1.732s	0.349s	0.184s	0.486s	0.349s	0.006s	26f/s
640×480	1.744s	0.961s	0.236s	0.491s	0.961s	0.009s	17f/s

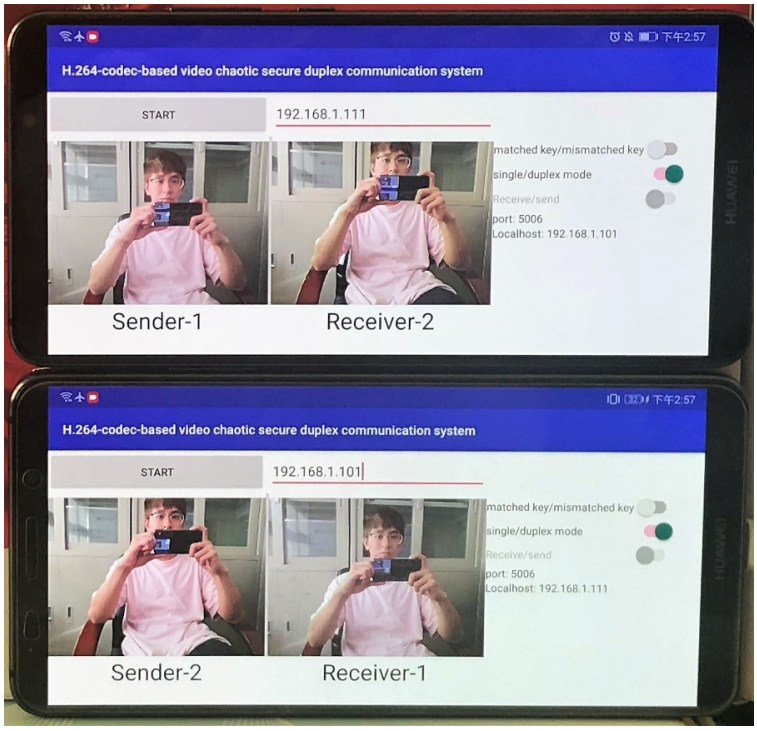

Fig. 5. Hardware implementation results with matched keys

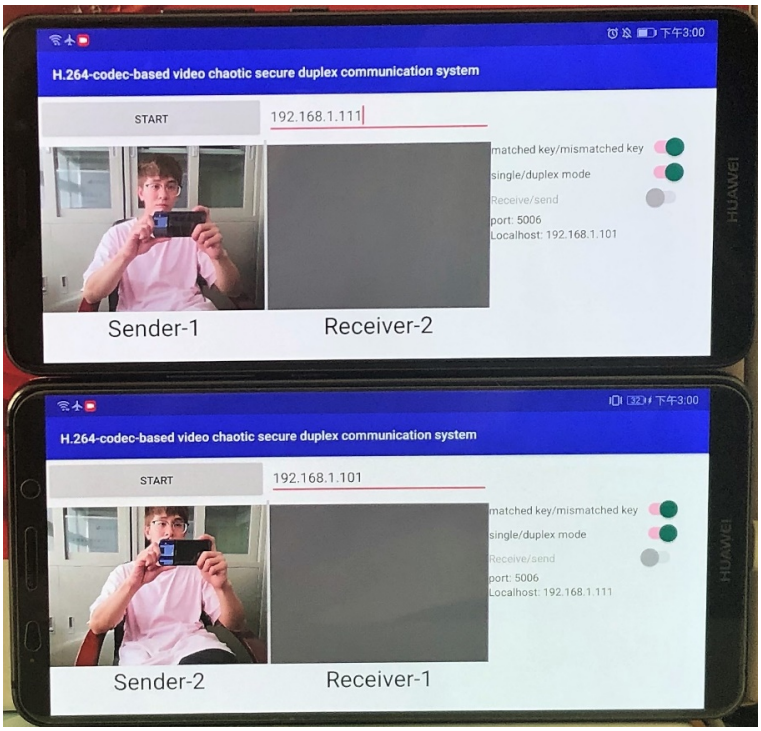

Fig. 6. Hardware implementation results with mismatched key $|\Delta a_{31}| = 10^{-6}$

6. Conclusions

Up to the present, the cryptanalysis of chaotic encryption algorithms has been mainly focused on open-loop chaotic cryptosystems and some of closed-loop feedback chaotic cryptosystems. However, it is rare to be reported for the chaotic self-synchronous chaotic stream cipher algorithms whose ciphertexts are fed back into the chaotic system. Although some reports suggest that some cryptanalysis methods (such as the single key decipher algorithm and the DCA-TSNCIC) can be used to decipher the original keys, these methods have some limitations. Meanwhile, some improved algorithms were also proposed to resist the existing attack methods, but these improved schemes may not be suitable for hardware implementation or resist other attack methods with stronger attack strengths. To deal with these problems, in this work, we proposed:

(1) An improved divide-and-conquer attack cryptanalysis method DCA-TMNCIC;

(2) An improved algorithm 3-D SCSCA-SMMR;

(3) An H.264-codec-based smartphone implementation scheme for video chaotic secure duplex communication.

With a stronger attack strength than the DCT-TSNCIC, the DCA-TMNCIC can make a more comprehensive security analysis for self-synchronous chaotic stream cipher algorithms. To resist the combination of the chosen-plaintext attack, known-plaintext attack, chosen-ciphertext attack and DCA-TMNCIC, the sinusoidal modulation, multiplication, modulo operation and round down operation are included in the improved algorithm. Compared with previously published works, listed in Table 1, the security of the self-synchronous chaotic stream cipher algorithms has been significantly improved. The improved chaotic encryption algorithm was then used in an H.264-codec-based video chaotic secure duplex communication system, and it is also convenient for smartphone implementations. The hardware implementation results have verified the feasibility of the proposed approach.

Acknowledgments

This work was supported by the National Key Research and Development Program of China (No.2016YFB0800401), the National Natural Science Foundation of China (No. 61532020, 61671161).

References

Arslan, S. & Junaid, S. [2018] "Novel image encryption cryptosystem based on binary bit planes extraction and multiple chaotic maps," *The European Physical Journal Plus* **133**, 331.

Chen, P., Yu, S., Chen, B., Xiao, L. & Lü, J. [2018a] "Design and SOPC-based realization of a video chaotic secure communication scheme," *International Journal of Bifurcation and Chaos* **28**, 1850160.

Chen, S., Yu, S., Lü, J., Chen, G. & He, J. [2018b] "Design and FPGA-based realization of a chaotic secure video communication system," *IEEE Transactions on Circuits and Systems for Video Technology* **28**, 2359–2371.

Dou, Y., Liu, X., Fan, H. & Li, M. [2017] "Cryptanalysis of a DNA and chaos based image encryption algorithm," *Optik* **145**, 456–464.

Elhosany, H. M., Hossin, H. E., Kazemian, H. B. & Faragallah, O. S. [2012] "Chaotic encryption of images in the fractional Fourier transform domain using different modes of operation," *2012 29th National Radio Science Conference (NRSC)* (IEEE), pp. 223–235.

Gan, Q., Yu, S., Li, C., Lü, J., Lin, Z. & Chen, P. [2017] "Design and ARM-embedded implementation of a chaotic map-based multicast scheme for multiuser speech wireless communication," *International Journal of Circuit Theory and Applications* **45**, 1849–1872.

Hua, Z. & Zhou, Y. [2016] "Image encryption using 2D Logistic-adjusted-Sine map," *Information Sciences* **339**, 237–253.

Jain, A. & Rajpal, N. [2016] "A robust image encryption algorithm resistant to attacks using DNA and chaotic logistic maps," *Multimedia Tools and Applications* **75**, 5455–5472.

Jolfaei, A., Wu, X.-W. & Muthukkumarasamy, V. [2015] "On the security of permutation-only image encryption schemes," *IEEE Transactions on Information Forensics and Security* **11**, 235–246.

Li, C., Lin, D., Lü, J. & Hao, F. [2018a] "Cryptanalyzing an image encryption algorithm based on autoblocking and electrocardiography," *IEEE Multimedia* **25**, 46–56.

Li, C. & Lo, K.-T. [2011] "Optimal quantitative cryptanalysis of permutation-only multimedia ciphers against plaintext attacks," *Signal Processing* **91**, 949–954.

Li, C., Zhang, Y. & Xie, E. Y. [2019] "When an attacker meets a cipher-image in 2018: A year in review," *Journal of Information Security and Applications* **48**, 102361.

Li, M., Lu, D., Wen, W., Ren, H. & Zhang, Y. [2018b] "Cryptanalyzing a color image encryption scheme based on hybrid hyper-chaotic system and cellular automata," *IEEE Access* **6**, 47102–47111.

Lin, Z., Wang, G., Wang, X., Yu, S. & Lü, J. [2018a] "Security performance analysis of a chaotic stream cipher," *Nonlinear Dynamics* **94**, 1003–1017.

Lin, Z., Yu, S., Feng, X. & Lü, J. [2018b] "Cryptanalysis of a chaotic stream cipher and its improved scheme," *International Journal of Bifurcation and Chaos* **28**, 1850086.

Lin, Z., Yu, S., Lü, J., Cai, S. & Chen, G. [2015] "Design and ARM-embedded implementation of a chaotic map-based real-time secure video communication system," *IEEE Transactions on Circuits and Systems for Video Technology* **25**, 1203–1216.

Matthews, R. [1989] "On the derivation of a "chaotic" encryption algorithm," *Cryptologia* **13**, 29–42.

Niu, Y., Zhang, X. & Han, F. [2017] "Image encryption algorithm based on hyperchaotic maps and nucleotide sequences database," *Computational Intelligence and Neuroscience* **2017**.

Niyat, A. Y., Moattar, M. H. & Torshiz, M. N. [2017] "Color image encryption based on hybrid hyper-chaotic system and cellular automata," *Optics and Lasers in Engineering* **90**, 225–237.

Norouzi, B., Mirzakuchaki, S., Seyedzadeh, S. M. & Mosavi, M. R. [2014] "A simple, sensitive and secure image encryption algorithm based on hyper-chaotic system with only one round diffusion process," *Multimedia Tools and Applications* **71**, 1469–1497.

Özkaynak, F. [2018] "Brief review on application of nonlinear dynamics in image encryption," *Nonlinear Dynamics* **92**, 305–313.

Parvin, Z., Seyedarabi, H. & Shamsi, M. [2016] "A new secure and sensitive image encryption scheme based on new substitution with chaotic function," *Multimedia Tools and Applications* **75**, 10631–10648.

Shahzadi, R., Anwar, S. M., Qamar, F., Ali, M. & Rodrigues, J. J. [2019] "Chaos based enhanced RC5 algorithm for security and integrity of clinical images in remote health monitoring," *IEEE Access* **7**, 52858–52870.

Song, C. & Qiao, Y. [2015] "A novel image encryption algorithm based on DNA encoding and spatiotemporal chaos," *Entropy* **17**, 6954–6968.

Wen, H. & Yu, S. [2019] "Cryptanalysis of an image encryption cryptosystem based on binary bit planes extraction and multiple chaotic maps," *The European Physical Journal Plus* **134**, 337.

Wen, H., Yu, S. & Lü, J. [2019] "Breaking an image encryption algorithm based on DNA encoding and spatiotemporal chaos," *Entropy* **21**, 246.

Wu, C. W. & Chua, L. O. [1993] "A simple way to synchronize chaotic systems with applications to secure communication systems," *International Journal of Bifurcation and Chaos* **3**, 1619–1627.

Xu, M. & Tian, Z. [2019] "A novel image cipher based on 3D bit matrix and latin cubes," *Information Sciences* **478**, 1–14.

Yang, Y., Pan, Q., Sun, S. & Xu, P. [2015] "Novel image encryption based on quantum walks," *Scientific Reports* **5**, 1–9.

Ye, G. & Huang, X. [2015] "An image encryption algorithm based on autoblocking and electrocardiography," *IEEE Multimedia* **23**, 64–71.

Zhang, L. Y., Liu, Y., Wang, C., Zhou, J., Zhang, Y. & Chen, G. [2018] "Improved known-plaintext attack to permutation-only multimedia ciphers," *Information Sciences* **430**, 228–239.

Zhang, X., Wang, L., Zhou, Z. & Niu, Y. [2019] "A chaos-based image encryption technique utilizing hilbert curves and H-fractals," *IEEE Access* **7**, 74734–74746.

Zhang, Z. & Yu, S. [2019] "On the security of a latin-bit cube-based image chaotic encryption algorithm," *Entropy* **21**, 888.

Zhu, S., Zhu, C. & Wang, W. [2018] "A new image encryption algorithm based on chaos and secure hash SHA-256," *Entropy* **20**, 716.