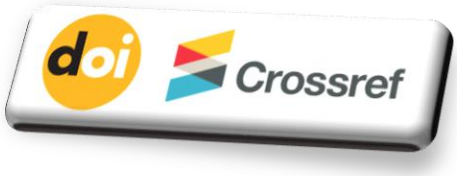
Journal, Advances in Mathematical & Computational Sciences
Vol. 8 No. 3, September, 2020
www.mathematics-computationaljournal.info



Journal of Advances in Mathematical & Computational Sciences
An International Pan-African Multidisciplinary Journal of the SMART Research Group
International Centre for IT & Development (ICITD)
Southern University Baton Rouge, LA, USA

https://www.isteams.net/socialinformaticsjournal
DOI: dx.doi.org/10.22624/AIMS/MATHS/ V8N3P4
CrossREF Member Listing - https://www.crossref.org/06members/50go-live.html

Signal with Blake2, Chacha20 and Poly1305

[1]Atiku, A.U. & [2]Sajoh, D.I.
[1,2]Department of Computer Science
Modibbo Adama University of Technology
Yola, Adamawa State, Nigeria
[1]ahmed.atiku@mautech.edu.ng, [2]disajoh@mautech.edu.ng

ABSTRACT

Messaging applications are now adopting the concept of end to end encryption. Signal uses the Signal Protocol which has no known vulnerabilities for its end to end encryption. The protocol was explained in this work and possibilities of improvements were explored. It was discovered that the protocol implementation uses the SHA-512 hash, AES in CBC mode for encryption and HMAC for authentication. These Cryptographic Primitives were replaced with more efficient alternatives, Blake2, Chacha20 and Poly1305 respectively. An application was created to test the running time of the proposed cryptographic primitives against the existing one. From the results, Blake2b performs better than SHA-512 when hashing keys, short text, and 1.4 MB file. Similarly, Chacha20/Poly1305 performs better than AES/CBC/HMAC-sha256 in both encrypting and decrypting short text as well as 1.4 MB file. The proposed changes were implemented in the Signal Application. Automatic test in the Signal application was used to ensure that changes made to the code did not cause any logic errors. To further test the application, it was installed on another phone and messages were sent. The security code was also verified.

.

Keywords: Signal, Blake2, Chacha20, Poly1305

Atiku, A.U. & Sajoh, D.I..(2020): Signal with Blake2, Chacha20 and Poly1305. Journal of Advances in Mathematical & Computational Sc. Vol.8, No. 3. Pp 37-48. DOI: dx.doi.org/10.22624/AIMS/MATHS/ V8N3P4. Available online at www.isteams.net/mathematics-computationaljournal.

1. INTRODUCTION

Signal is a messaging application developed by Open Whisper Systems. The application has end to end encryption that is built on a protocol called Signal Protocol, which was also developed by Open Whisper Systems[2]. The protocol is a combination of two protocols: Textsecure, which uses the concept of double ratchetting to ensure future secrecy and Redphone[8][10]. Signal protocol didn't get much attention in the research world until its integration to WhatsApp in 2015 [4]. [8] did a formal security analysis of the protocol focusing on the multistage Authenticated Key Exchange (AKE) protocol. They showed that the protocol is resistant to Unknown Keyshare Attacks and did not find any serious vulnerabilities.

Considering WhatsApp's large user base, a comparison of the protocol implementation was done in [10]. They identified the differences in the implementation and justified WhatsApp's decision to sacrifice security for usability in some parts of its implementation. The protocol is also used to provide end to end encryption in Facebook, Wire and Google Allo[15].

These applications, including WhatsApp and Signal, are used mostly in mobile devices which have limited resources. It is therefore important that the Signal Protocol is efficient. The focus of this paper is improving the efficiency of the protocol implementation in Signal using faster encryption algorithms without compromising security.

2. THE SIGNAL PROTOCOL

Asymmetric Keys Used

- *ika, IKA*: Alice's Identity key pair
- *eka, EKA*: Alice's Ephemeral key pair
- *rcka^i, RCKA^i*: Alice's ith Ratchet Key pair
- *ikb, IKB*: Bob's Identity key pair
- *spkb, SPKB*: A pair of Bob's Signed Pre-key
- *(opkb, OPKB)**: multiple pairs of Bob's one time pre-keys. A single one will be written without the*
- *rckb^i,RCKB ^{i}*: Bob's ith Ratchet Key

The keys written in lower case represent the private keys while those in upper case are the public keys.

Symmetric Keys used

- *rki*: The ith Root Key
- *cki*: The ith Chain Key
- *mki*: The ith Message Key

Other Abbreviations

- ECDH: Elliptic Curve Deffie Hellman key exchange
- DH(ka, KB): An ECDH using a private key belonging to Alice and the other Bob's public key
- $Sig_k(M)$: Signing a message M with a private key
- $Ver_{KB}(Sign)$: Verify Bob's Signature with his public key
- AD: Associated Data
- AEAD: Authenticated Encryption with Associated Data. This is a set-up that provides both encryption and MAC of a message with its AD
- $E_k(M,AD)$: Encrypting a message M along with its AD using a key *k* in an AEAD set-up
- $D_k(C,AD)$: Decrypting a message C along with its AD using a key *k* in an AEAD set-up
- *KDF(sec)*: A key derivation function that takes in a secret *sec* and uses it to derive two new secrets
- *KDF2(sec1, sec2)*: A key derivation function that takes in a two secrets *sec1, sec2* and uses it to derive two new secrets

The Signal Protocol description can be divided into three stages: Key Registration, Trust Establishment and Messaging.

Journal, Advances in Mathematical & Computational Sciences
Vol. 8 No. 3, September, 2020
www.mathematics-computationaljournal.info

2.1 Key Registration

At install time, Bob most first register his phone number which will be used as his identity. To confirm that he is the actual owner of that number, the server sends an SMS (Short Message Service) with a confirmation code to the phone number entered. Bob now enters this confirmation code in the Application. If verified, the phone number will be registered to the server and Bob will generate some elliptic curve key pairs and send the public keys to the server as shown in Fig.1. Once the server registers Bob's details.

Other users can communicate with Bob using the protocol. The use of confirmation code is not specified in the Signal Protocol, but it is used at implementation stage by the messaging application. The Signal Protocol is based on a Trust On First Use (TOFU) scheme and it does not provide any cryptographic guarantee that an attacker cannot send his own elliptic curve keys to the server with Bob's identity[13]. However, the protocol provides a means of authentication with a security code generated from the public keys as explained in section 2.2. The keys sent to the server by Bob are IKB, SPKB and OPKB*. IKB is a permanent key and it only changes when Bob reinstalls the application or changes his device. SPKB is periodically changed and sent to the server together with the signature SignB. When the server is running out of OPKB, it notifies Bob and he sends more OPKB*.

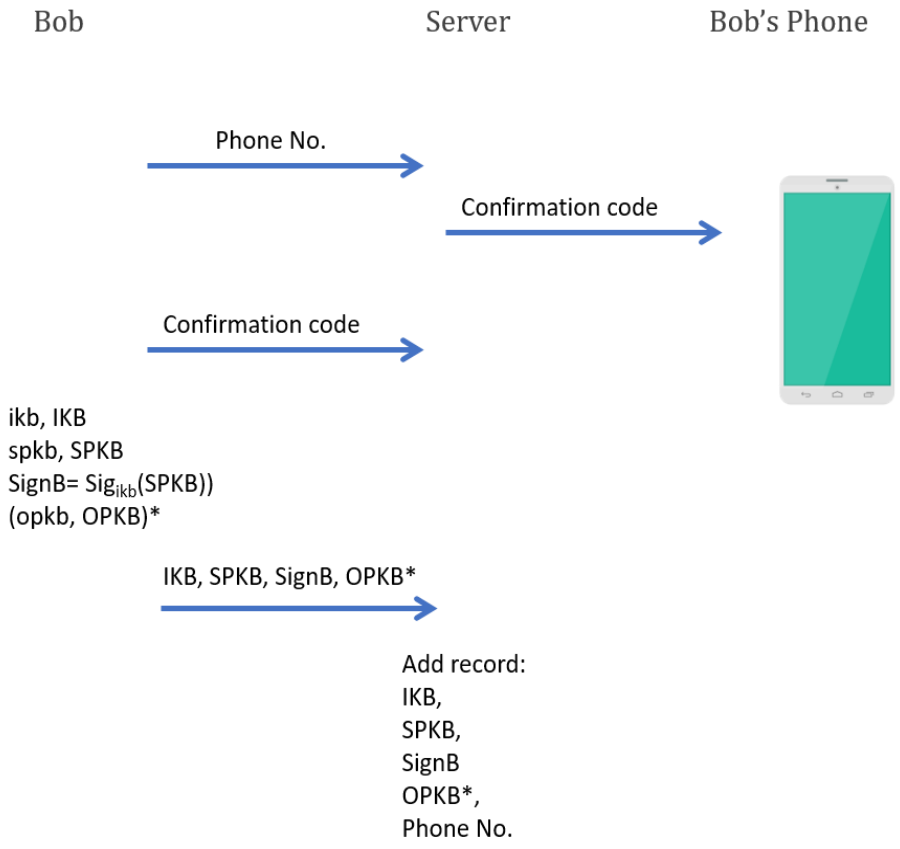


Figure 1: The Registration Process

2.2 Key Exchange and Trust Establishment

If Alice tries to start a conversation with Bob, she initiates the key exchange as shown in Fig. 2 when she sends the first message to Bob. Once Alice initiates the sending process, the server sends IKB, SPKB, SignB, and one OPKB if available. After receiving the public keys, she verifies the signature SignB with IKB. If the verification fails, she aborts the protocol. After successfully verifying SignB, she generates an ephemeral key pair (eka, EKA) and computes DH1, DH2, DH3 and DH4. If Bob ran out of OPKB and none was sent by the server, she will not compute DH4. After computing the three or four DH, she enters the concatenation of all the DH into a KDF to generate the keys ck and mk. mk is used to encrypt the message M using an AEAD with additional data AD and ck is used in the ratcheting process. She then generates a ratcheting key pair (rcka, RCKA) and sends the cipher text C, RCKA and EKA to the server which then forwards it to Bob. Once Bob is online, he receives EKA and C and an indication of the OPKB that was sent to Alice.

Then, he computes DH1, DH2, DH3 and DH4. If an OPKB was not sent to Alice, he will not compute DH4 and will generate more (opk, OPK) pairs and send to server. After computing the three or four DH, he also generates ck and mk using a KDF and AD the way Alice did and use them to decrypt C using the AEAD scheme. A security code will be created by hashing IKA and IKB and Alice and Bob can use it for authentication.

At this stage, a session has been created between Alice and Bob. Bob can now delete the (opk, OPK) pair used and Alice will delete the (eka, EKA) as they will not be used again. They can also delete the DH. Since both Alice and Bob register their public keys with the server at install time as shown in Fig. 1, Alice can get all the information required to send a message to Bob even if Bob is offline and when Bob gets online, he can read her message and send a reply even if she is offline. This is what makes the Trust Establishment asynchronous.

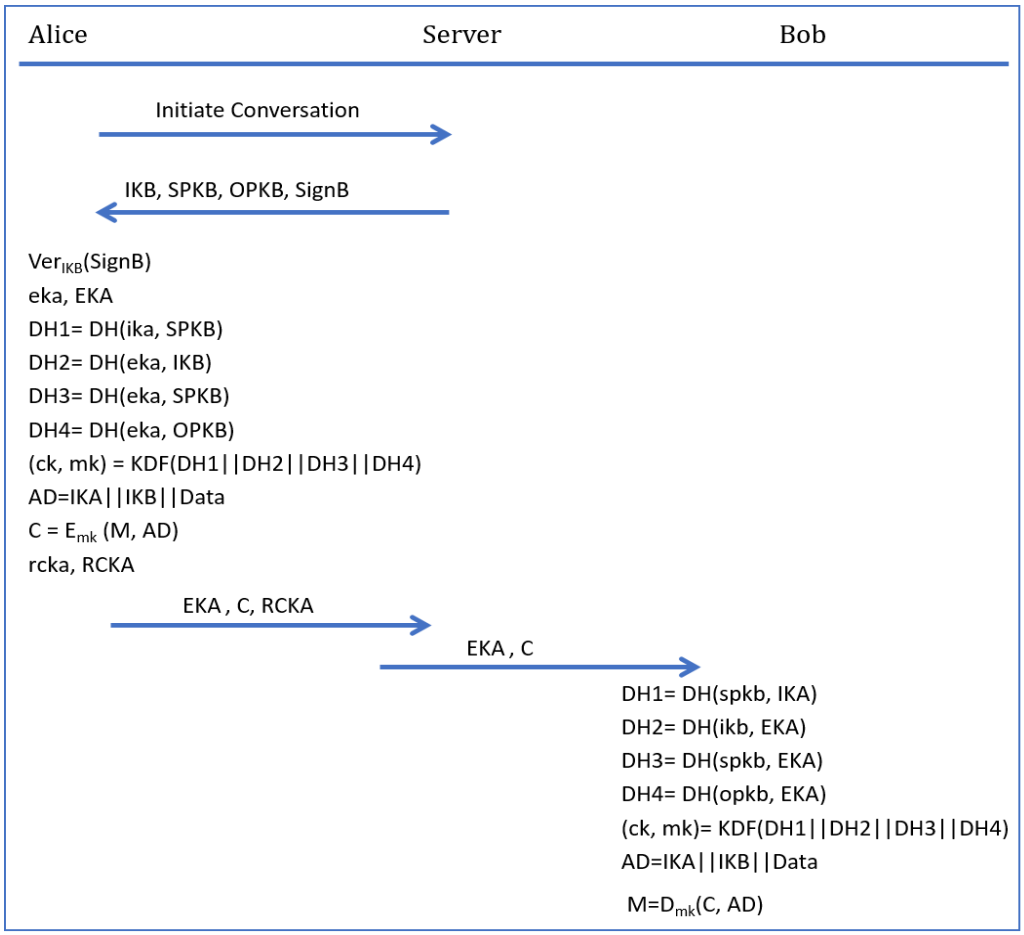


Figure 2: The Signal Key Exchange

2.3 Messaging

Once Alice and Bob complete the key exchange and have shared secret (ck0, mk0), they start a process called Double Ratcheting. The Double Ratcheting protocol specified in Signal documentation[14] has two ratcheting protocols; Asymmetric and Symmetric.

When sending the first message after key exchange, Alice also sends her first RCKA. When Bob wants to reply to Alice's message, he generates his (rckb, RCKB) pair and performs a DH with RCKA. He uses that DH and the shared secret rk0 to generate a new rk1 and ck1 using KDF2. The use of the DH to generate new keys is the Asymmetric ratchet step. The new ck1 is used to generate a new pair of secrets ck2 and mk1. This is the Symmetric ratchet step. Bob now uses the new secret m1 to encrypt his reply and sends the cipher text together with RCKB. Alice follows the same procedure when she replies to Bob. This way, they both take turns in introducing a new DH into the protocol, hence achieving Backward Secrecy. Fig.3 gives further explanation of this process.

Journal, Advances in Mathematical & Computational Sciences
Vol. 8 No. 3, September, 2020
www.mathematics-computationaljournal.info

If Bob has not replied the message sent by Alice and she sends another message. Asymmetric ratchet cannot be applied since there is no new DH. In that case, she continues applying Symmetric ratchet for each message she sends until Bob replies and an Asymmetric ratchet is performed. Bob also does the same when sending consecutive messages. Fig. 4 shows the process of sending consecutive messages.

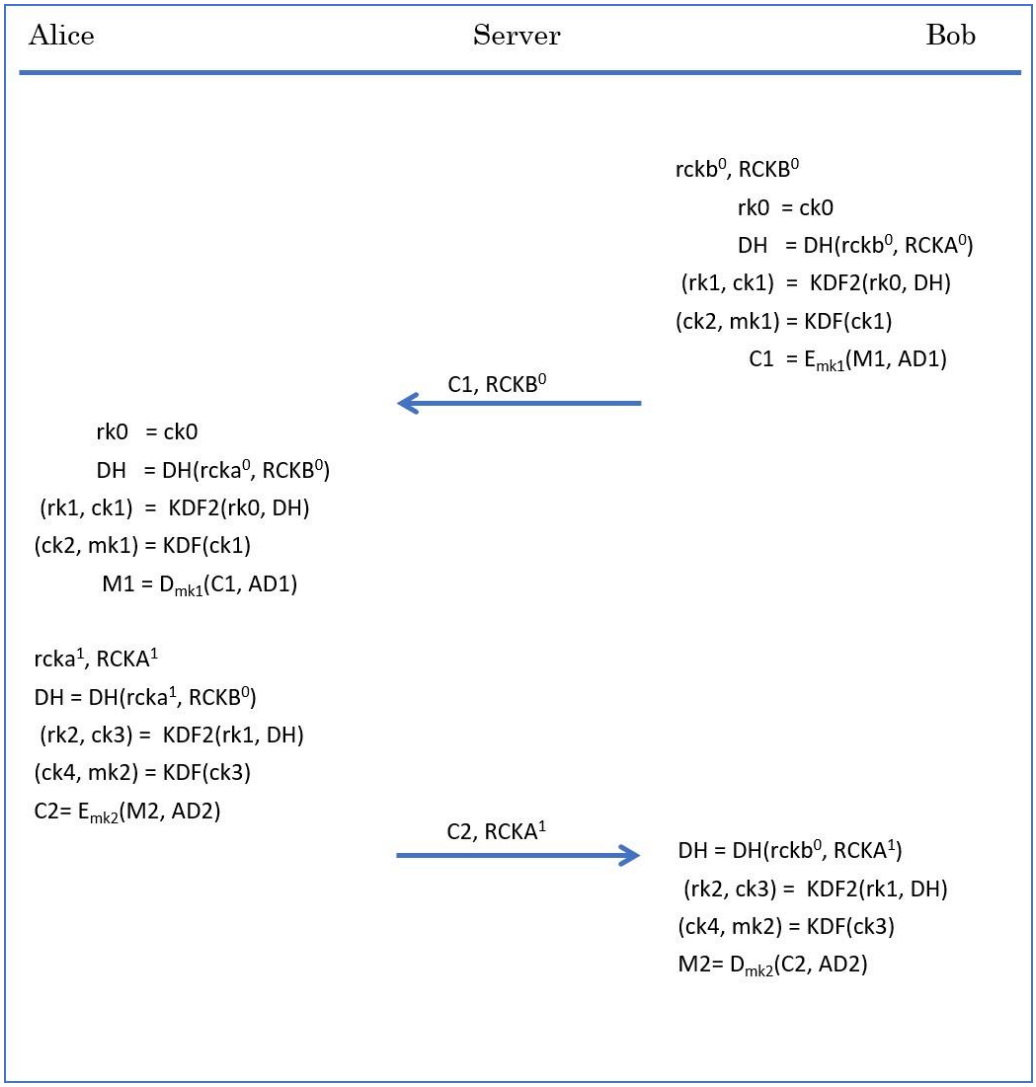


Figure 3: The Ratcheting Protocol: This shows what happens when Alice and Bob take turns in sending messages.

Once a ck or mk is used, it can be deleted. The rk is only deleted when a new rk is generated from an asymmetric ratchet step. The secrets used by Alice in encryption form her sending chain while that of decryption is her receiving chain. Alice's sending chain is therefore equivalent to Bob's receiving chain and vice versa. A new sending and receiving chain is formed whenever an asymmetric ratchet step is performed. The message number in the sending chain is included in the message to track out of order messages.

3. CRYPTOGRAPHIC PRIMITIVES

The implementation of the Signal Protocol in Signal uses curve25519 Elliptic Curve Digital Signature Algorithm for signing and verifying messages. The hash function used is SHA-512. AES in CBC mode with HMAC-Sha256 authentication is used as AEAD and the key derivation functions are HMAC based. These cryptographic primitives are all considered to be secure and the Signal protocol has no known major vulnerabilities[9]. This work focuses on improving the efficiency of the Signal Protocol using some efficient state-of-the-art cryptographic primitives suggested in [3] to replace the existing primitives. The primitives considered are Blake2 hash function and Chacha20 with Poly1305 AEAD.

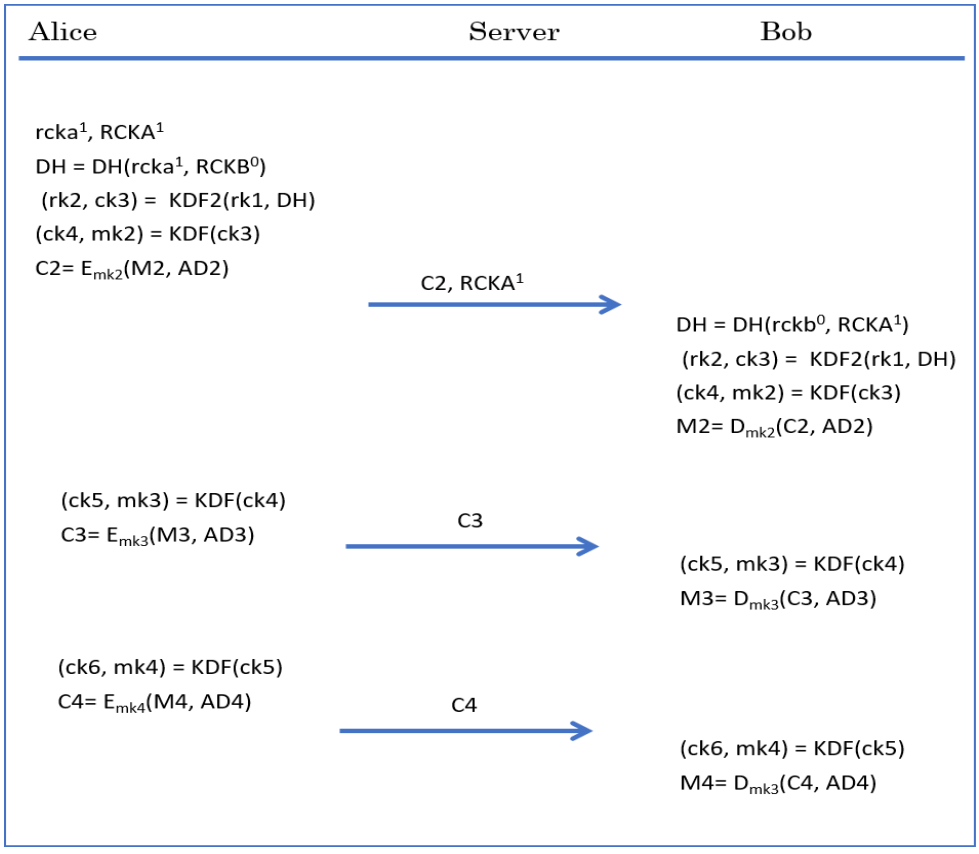


Figure 4: The Ratcheting Protocol: This shows what happens when Alice and Bob take turns in sending messages.

Blake2 is a successor of Blake, which is one of the SHA-3 finalist [6]. It is a reliable hash function considered to be even faster than MD5 hash if properly implemented[6]. It is also resistant to different forms of attack[11][5]. ChaCha20 is a fast stream cypher which is sometimes used as a block cipher[7]. It uses Addition Rotation and Xor (ARX) which is more CPU friendly than the mixed columns and mixed rows used in AES[7]. Poly1305 is a fast Message Authentication Code [7]. It can be used together with Poly1305 in an AEAD. Chacha20/Poly1305 AEAD is already used in TLS[7]. It is also resistant to different attacks[12].

An application was created to test the running time of the proposed cryptographic primitives against the existing one. The application takes the inbuilt Java implementation of SHA-512 as used by Signal and measures the time it takes to take to hash a public key, a short text and a 1.4 MB file. It then takes the Blake2b implementation and measures the time it will take to perform the same hash. The application also measures the running time of performing encryption and decryption of a short text and a 1.4 MB files using AES/CBC/HMAC-sha256 and Chacha20/Poly1305. The tests were performed ten times each and the average time was measured.

From the results, the average execution time of hashing keys with Blake2b was 67,880ns while that of SHA-512 was 90,647ns which is higher. The average execution time of Blake2b when hashing a short text was 59,156ns and that of SHA-512 was 84,956ns which is also higher. The average execution time of hashing a 1.4 MB file using Blake2b was 8,019,587 and it is also than that of SHA-512 which was 9,309,547ns (see table 1).

Table 1: Comparison of the efficiency of Blake2 and SHA-512

		Time (ns)	
		Blake2b	Sha512
Hash Public keys	Test 1	85,375	145,092
	Test 2	57,383	106,369
	Test 3	55,984	101,238
	Test 4	80,710	90,974
	Test 5	55,051	61,115
	Test 6	61,583	62,982
	Test 7	55,984	63,449
	Test 8	85,841	101,703
	Test 9	55,051	52,252
	Test 10	85,842	121,298
	Average:	67,880	90,647

		Time (ns)	
		Blake2b	Sha512
Hash Short Text	Test 1	57,384	95,173
	Test 2	56,451	79,777
	Test 3	57,850	86,309
	Test 4	65,315	84,909
	Test 5	56,450	92,840
	Test 6	55,983	93,773
	Test 7	56,450	87,241
	Test 8	55,517	66,714
	Test 9	56,450	72,312
	Test 10	73,712	90,507
	Average:	59,156	84,956

		Time (ns)	
		Blake2b	Sha512
Hash 1.4MB File	Test 1	8,698,950	9,703,860
	Test 2	7,377,266	8,418,098
	Test 3	9,318,504	9,631,081
	Test 4	7,842,398	9,799,032
	Test 5	8,308,930	9,521,446
	Test 6	7,812,074	9,544,306
	Test 7	7,192,986	8,149,842
	Test 8	7,815,806	9,660,005
	Test 9	8,064,934	8,754,468
	Test 10	7,764,021	9,913,332
	Average:	8,019,587	9,309,547

Table 2: Comparison of the efficiency of encrypting and decrypting a short text using AES/CBC/HMAC and Chacha20/Poly1303 AEAD schemes

		Time (ns)			
		Encryption		Decryption	
		ChaCha20 /Poly1305	AES/CBC/ HMAC	ChaCha20/ Poly1305	AES/CBC/ HMAC
Short Text	Test 1	1,919,311	2,453,024	1,785,884	2,006,086
	Test 2	1,586,208	1,806,877	1,592,740	1,679,980
	Test 3	1,873,591	2,238,419	1,768,156	1,941,705
	Test 4	1,995,356	1,891,787	1,653,388	1,765,822
	Test 5	2,176,370	2,881,300	1,590,873	1,679,048
	Test 6	2,024,748	1,864,727	1,626,797	1,749,027
	Test 7	1,667,384	1,968,298	1,623,997	1,678,581
	Test 8	1,662,252	1,812,476	1,611,867	1,708,439
	Test 9	1,653,338	2,039,677	1,565,214	2,226,755
	Test 10	1,706,573	1,929,576	2,405,437	1,920,711
	Average:	1,826,513	2,088,616	1,722,435	1,835,615

Table 3: Comparison of the efficiency of encrypting and decrypting a 1.4 MB File using AES/CBC/HMAC and Chacha20/Poly1303 AEAD schemes

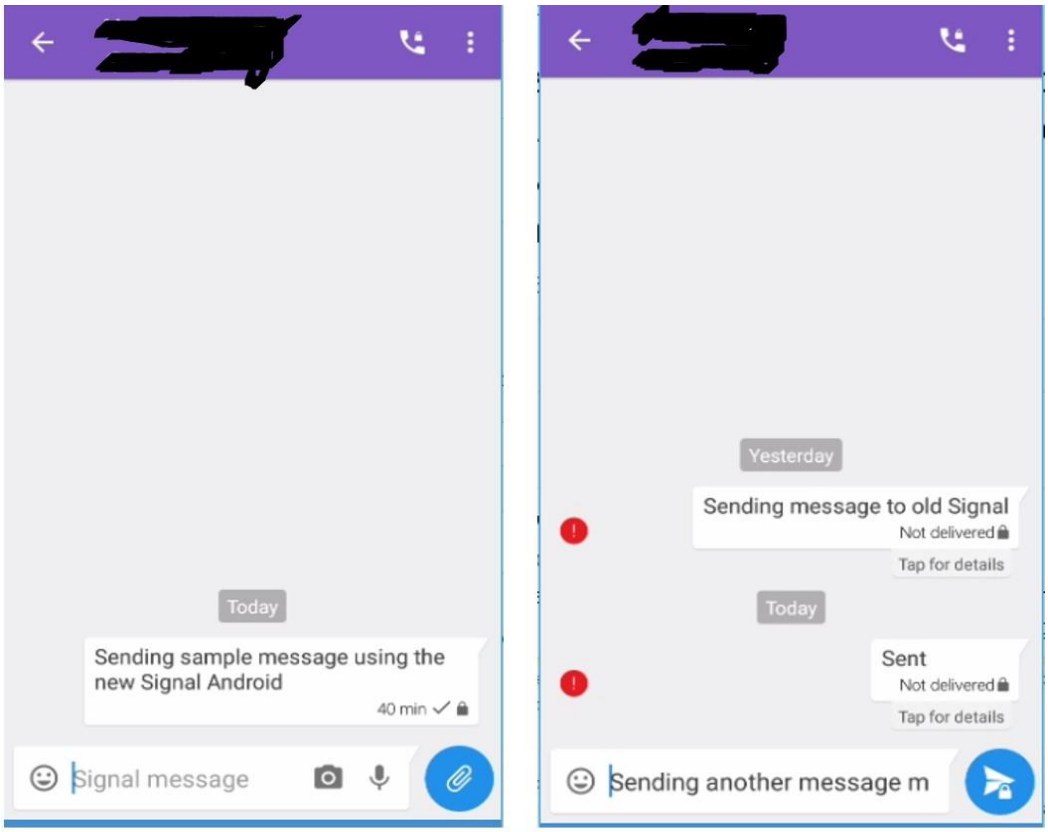
		Time (ns)			
		Encryption		Decryption	
		ChaCha20 /Poly1305	AES/CBC/ HMAC	ChaCha20/ Poly1305	AES/CBC/ HMAC
1.4 MB File	Test 1	16,707,434	20,807,781	15,347,494	18,422,405
	Test 2	15,758,042	22,614,191	14,479,278	17,924,149
	Test 3	17,679,686	22,989,283	18,644,940	21,248,653
	Test 4	14,727,474	19,142,729	14,880,962	18,164,412
	Test 5	14,856,703	26,969,266	14,511,469	18,050,578
	Test 6	14,153,639	18,667,334	14,197,960	17,693,215
	Test 7	16,965,426	21,675,064	14,285,201	18,048,712
	Test 8	16,938,367	20,888,491	14,661,226	19,614,392
	Test 9	14,808,183	18,671,998	14,775,526	18,227,394
	Test 10	15,602,220	19,034,494	14,964,471	19,118,470
	Average:	15,819,717	21,146,063	15,074,853	18,651,238

As shown in table 2, the average execution times for encrypting and decrypting a short text using Chacha20/Poly1305 were 1,826,513ns and 1,722,435ns respectively. Whereas, the average execution times for encrypting and decrypting same text using AES/CBC/HMAC-sha256 were 2,088,616ns and 1,835,615ns respectively. For 1.4MB file encryption and decryption, the average execution times for Chacha20/Poly1305 were 15,819,717ns and that of AES/CBC/HMAC-sha256 were 18,651,238ns. Table 3 provides more information on the speed of file encryption and decryption. In both cases, both encryption and decryption times in Chacha20/Poly1305 were lower than the corresponding times in AES/CBC/HMAC-sha256.

4. THE IMPROVED SIGNAL

The cryptographic primitives were changed in the android version of the Signal application available in Open Whisper System's GitHub page[1]. The automatic tests in the Signal application was used to ensure that changes made to the code did not cause any logic errors. To further test the application, it was installed on another phone and messages were sent. The security code was also verified.

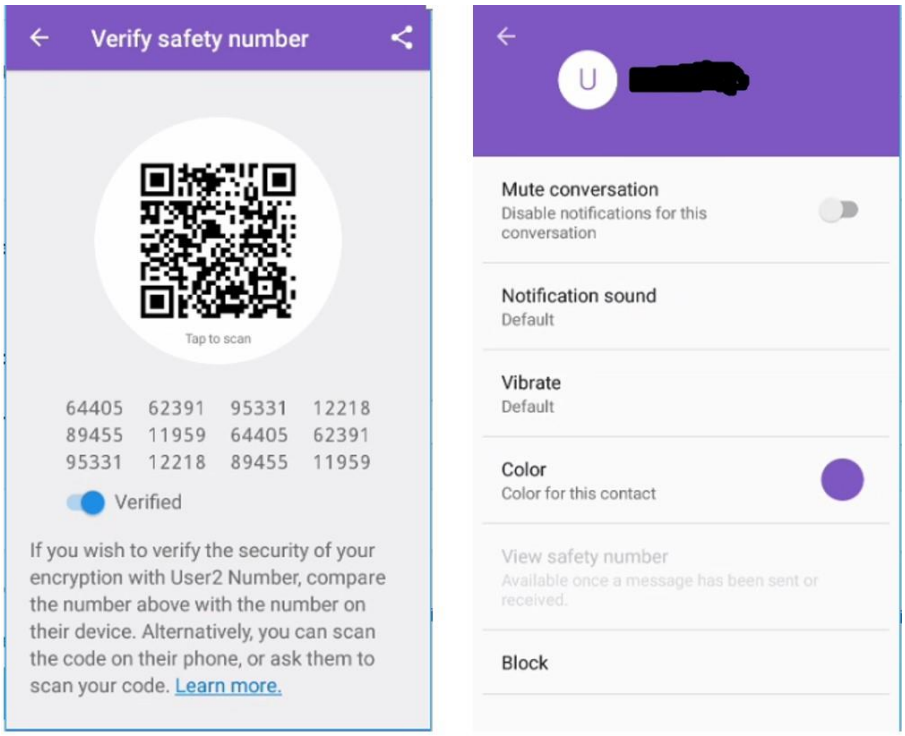
When the improved application was used to send a message to a user using the old application, the messages failed to deliver. This is expected because the key exchange will fail since the second user was using SHA-512 in $Sig_{ikb}(SPKB)$ and the first user is using Blake2 in the $Ver_{IKB}(SignB)$. Figure 5 shows the result of sending messages with the improved application.



Message successfully sent using the new Signal Android

Message sent to original Signal android with the new Application failed to deliver

Figure 5: The result of sending messages using the new Signal Android.



Security code successfully generated and verified using the new Application

Security code cannot be generated because there is no successfully sent message from the new application to the original one

Figure 6: The result of verifying security code using the new Signal Android.

The verification of the security code between the two users using the improved application was also successful, but verification could not be performed with the user using the old application since the key exchange failed and the security code was not generated. Figure 6 shows the result of verifying security code using the improved application.

5. CONCLUSION

In this work, it was shown that the implementation of Blake2 hash function on short text and files is more efficient than that of sha512. Chacha20 with Poly1305 AEAD was also proved to be faster than AES in CBC mode with HMAC-Sha256. Since the Signal Protocol is independent of any cryptographic algorithm or hash function, replacing the hash function and cryptographic algorithm did not require any alteration to the protocol itself. The cryptographic primitives can always be replaced with more secure ones if broken in the future.

REFERENCES

[1] Open whisper systems. https://github.com/WhisperSystems.

[2] Signal. https://signal.org/#page-top. Accessed: 2018-10-06.

[3] Wireguard: Fast, modern, secure vpn tunnel. https://www.wireguard.com/.

[4] Jo¨el Alwen, Sandro Coretti, and Yevgeniy Dodis. The double ratchet: Security notions, proofs, and modularization for the signal protocol. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 129–158. Springer, 2019.

[5] Jean-Philippe Aumasson, Willi Meier, Raphael C-W Phan, and Luca Henzen. Blake2. In *The Hash Function BLAKE*, pages 165–183. Springer, 2014.

[6] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, and Christian Winnerlein. Blake2: simpler, smaller, fast as md5. In *International Conference on Applied Cryptography and Network Security*, pages 119–135. Springer, 2013.

[7] Daniel J Bernstein. Chacha, a variant of salsa20. In *Workshop Record of SASC*, volume 8, pages 3–5, 2008.

[8] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 451–466. IEEE, 2017.

[9] Katriel Cohn-Gordon, Cas Cremers, Benjamin Dowling, Luke Garratt, and Douglas Stebila. A formal security analysis of the signal messaging protocol. In *Security and Privacy (EuroS&P), 2017 IEEE European Symposium on*, pages 451–466. IEEE, 2017.

[10] D I Dajoh, A U Atiku, and R S Naibi. Secure messaging: Analysis of signal protocol implementation in whatsapp and signal. *Journal of Digital Inovations and Contemp Res. in Sc., Eng. and Tech.*, 6(3):63–72, 2018.

[11] Jian Guo, Pierre Karpman, Ivica Nikoli´c, Lei Wang, and Shuang Wu. Analysis of blake2. In *Cryptographers' Track at the RSA Conference*, pages 402–423. Springer, 2014.

[12] Kazuya Imamura, Kazuhiko Minematsu, and Tetsu Iwata. Integrity analysis of authenticated encryption based on stream ciphers. *International Journal of Information Security*, pages 1–19, 2016.

[13] Moxie Marlinspike and Trevor Perrin. The x3dh key agreement protocol. *Open Whisper Systems Specifications: The X3DH Key Agreement Protocol*, 2016.

[14] Trevor Perrin and Moxie Marlinspike. The double ratchet algorithm. *GitHub wiki*, 2016.

[15] Nik Unger, Sergej Dechand, Joseph Bonneau, Sascha Fahl, Henning Perl, Ian Goldberg, and Matthew Smith. Sok: secure messaging. In *Security and Privacy (SP), 2015 IEEE Symposium on*, pages 232–249. IEEE, 2015.