DOI: 10.2507/32nd.daaam.proceedings.013

SECURING COMMUNICATION IN CHATTER APPLICATION

Zlatan Morić, Jure Pinjuh & Sanjin Kurelic

This Publication has to be referred as: Moric, Z[latan]; Pinjuh, J[ure] & Kurelic, S[anjin] (2021). Securing Communication in Chatter Application, Proceedings of the 32nd DAAAM International Symposium, pp.0090-0097, B. Katalinic (Ed.), Published by DAAAM International, ISBN 978-3-902734-33-4, ISSN 1726-9679, Vienna, Austria
DOI: 10.2507/32nd.daaam.proceedings.013

Abstract

Chatter is an Angular web application that offers registered users' ability to chat with other registered users. Users can chat with multiple users in a secure fashion. The main problem in designing a good backend for the Chatter application is dealing with message exchange security. In this paper, we will present a few security protocols and suggestions for exchanging secure messages. From single substitution ciphers (*Caesar* and *Atbash*) and multi-alphabet substitution ciphers (Vigenère and Playfair) to more complex like symmetric (XOR stream cipher and DES block ciphers) and asymmetric algorithms (using private and public key mechanisms with Diffie-Hellman key exchange and RSA algorithm with hash functions - SHA). We will present the pros and cons of using each algorithm by using man-in-the-middle attack and choose a secure asymmetric algorithm as the main security for exchanging messages at the end.

Keywords: secure message exchange; substitution cipher; asymmetric algorithm; hash; man-in-the-middle attack.

1. Introduction

Globally, the number of internet users increased from only 413 million in 2000 to over 3.4 billion in 2016. The one billion barriers were crossed in 2005. Every day over the past five years, an average of 640,000 people went online for the first time [1]. As more and more people got online more of them are becoming technically educated and more of them care about their personal data. Personal data can be defined as any information relating to an identified or identifiable natural person ('data subject'); an identifiable natural person can be identified, directly or indirectly, in particular by reference to an identifier such as a name, an identification number, location data, an online identifier or to one or more factors specific to the physical, physiological, genetic, mental, economic, cultural or social identity of that natural person [2].

People want to protect their personal data and activity on the Internet. In recent times more and more people are protesting big tech companies like Facebook, Google, Microsoft, etc. They are moving from Facebook Messenger and WhatsApp because of the new usage policy to more secure platforms like Telegram or Signal. Secure communication between two entities become the most important feature of chatting application – protecting both entities from the hacker (man-in-the-middle) and from the company itself. Companies that are offering chatting applications should not install a backdoor in communication nor they should be man-in-the-middle by offering secure communication only between user and server. Choosing the right security mechanism for message exchange can be tricky if it’s not done properly.

Man-in-the-middle attack is one of the most popular and challenging threats in network security and there is a large body of research dedicated to the detection and analysis of different forms of these. A man-in-the-middle attack is defined as an attack in which the intruder can read and write messages communicated between two parties without either party being conscious of this fact [3]. Chatter is a chat web application built in Angular that aims to provide secure communication between registered users. Security is implemented by using the Forge framework for communication and the Keypair framework for generating RSA keys.

2. Classic cryptography

Securing message communication between two entities has been in the focus of people for more than 4000 years. During the war or trade, people used hidden and secret writing so enemy or competition doesn't reveal their plans. In past, cryptography was almost exclusively used by militaries and governments. Cryptography probably began in or around 2000 B.C. in Egypt, where hieroglyphics were used to decorate the tombs of deceased rulers and kings. These hieroglyphics told the story of the life of the king and proclaimed the great acts of his life. As time went by, these writings became more and more complicated, and eventually, the people lost interest in deciphering them [4].

In past, the most popular cipher was the substitution algorithm. Substitution is a cipher in which each occurrence of a plaintext symbol is replaced by a corresponding ciphertext symbol to generate the ciphertext. The key for such a cipher is a table of the correspondence or a function from which the correspondence is computed. Gary Knight uses the following classification for substitution algorithms: mono-alphabetic (replacing one character with ciphertext), polyalphabetic (each plaintext letter has defined multiple substitutions), poly-graphic (substitution in groups of text), and fractionating (ciphertext depends on multiple plaintext characters).

2.1. Mono-alphabetic substitution

Caesar cipher is probably the most famous mono-alphabetic substitution cipher, used by Julius Caesar. Every letter is shifted by a fixed number of spaces to the left or the right in the alphabet. In this cipher key is the number of characters that is shifting. For example, the word "Chatter" shifted by 4 is: "Glexxiv".

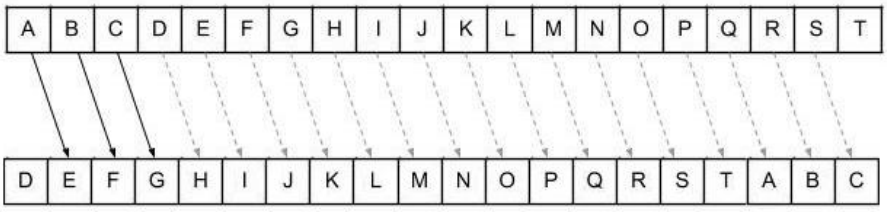

Fig. 1. Caesar cipher shifted by 4 characters

Caesar cipher can easily be implemented in JavaScript by using pure ASCII codes:

```
// Go through each character
for (var i = 0; i < str.length; i++) {
  // Get the character we'll be appending
  var c = str[i];

  // If it's a letter...
  if (c.match(/[a-z]/i)) {
    // Get its code
    var code = str.charCodeAt(i);

    // Uppercase letters
    if (code >= 65 && code <= 90) {
      c = String.fromCharCode(((code - 65 + amount) % 26) + 65);
    }

    // Lowercase letters
    else if (code >= 97 && code <= 122) {
      c = String.fromCharCode(((code - 97 + amount) % 26) + 97);
    }
  }
```

The problem with Caesar cipher is that it can be decrypted easily. A simple decrypt algorithm could shift all letters until it gets valid words. This can also be done interactively with an attacker who is checking the validity of sentences. The problem with Caesar cipher is also how to exchange the key (number of characters that is shifting). Hardcoding it in the application is even less secure because that information compromises all chats on the platform.

Atbash is a variation of the mono-alphabetic substitution algorithm which supplement the first letter of the alphabet to the last, the second letter to the second last, etc (Fig. 2). Using this cipher, we get a less obvious cipher technique, but once the attacker detects the algorithm, it's not possible to fix it (without changing the encryption algorithm). This is the opposite Caesar cipher with hardcoded key and security is only in hiding algorithm that application uses. Word “Chatter” using this algorithm would be: “xszggvi”.

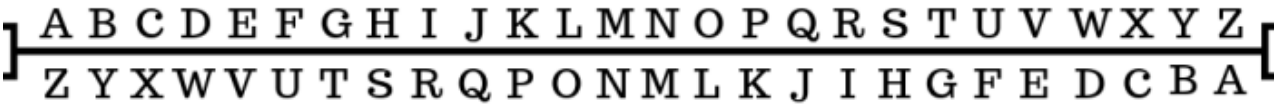

Fig. 2. Atbash cipher for the English alphabet

2.2. Poly-alphabet substitution

The statistical attack on the mono-alphabetic substitution cipher can be carried out because the key defines a fixed mapping that is applied letter-by-letter to the plain text. Such an attack could be thwarted by using polyalphabetic substitution cipher [5]. The most famous polyalphabetic substitution algorithm is Vigenère cipher. Vigenère cipher also known as a polyalphabetic shift cipher works by applying several independent instances of the shift cipher in sequence (Caesar cipher) [5]. For this cipher user must define the key that is used for encryption of the message. If a key is shorter than the length of the message, then the key is repeated (example: "key" becomes "keykeykey" for plaintext that has a length of nine characters). Vigenère cipher use Tabula Recta (Fig. 3) for encryption/decryption. Word "Chatter" with key "Chatter" by using Vigenère cipher becomes "eoammii".

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
A	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
B	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A
C	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B
D	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C
E	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D
F	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E
G	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F
H	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G
I	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H
J	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I
K	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J
L	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K
M	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L
N	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M
O	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N
P	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
Q	Q	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
R	R	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
S	S	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
T	T	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S
U	U	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
V	V	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
W	W	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
X	X	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
Y	Y	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X
Z	Z	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y

Fig. 3. Tabula Recta with annotated letters for encryption/decryption of word "Chatter"

The problem with Vigenère cipher is the length of the key. If the attacker finds the length of the key then she/he could easily decrypt text by using Tabula Recta and common attacks on shift cipher algorithms. The length of the key could be found by observing patterns in the ciphertext. In chat applications, people usually start a conversation with "Hi". Smaller messages could also contain “Ok”, “See you”, “Bye” and similar. Taking this into consideration, the attacker could easily find the start of the key, but not the length of it.

At first, this could be good for chatting applications, but users who like to type more words on chat or for example students who copy/paste a chunk of text from some literature could lead to breaking of this algorithm. By observing common words, like "the", the attacker can find the length of key, and break the algorithm by using techniques from mono-alphabetic substitution attacks.

2.3. Poly-graphic and fractionating substitution

Poly-graphic substitution adds more complexity to polyalphabetic substitution by doing encryption/decryption on blocks of text. One of the examples of poly-graphic substitution is Playfair cipher that encrypts pairs of letters instead of single letters. The frequency analysis of bigrams is possible, but more difficult with 600 possible bigrams rather than the 26 possible monograms. Fractionating substitution is a method of splitting letters so that each plaintext letter is represented by two or more symbols. This makes the message harder to break, however, the attacker could break it just like mono-alphabetic substitution by grouping letters in pairs (or diagraphs).

Plaintext Alphabet	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Ciphertext Alphabet	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26

Table 1. Example of fractionating substitution

Both these types of substitution make more complexity in breaking the message, but they are not enough for modern computers. By observing repetition patterns in the ciphertext and changing some letters and observing changes in cipher text, a hacker can easily break these algorithms and find cipher key or plain text value.

3. Modern cryptography

None of the mentioned security algorithms satisfies the premise of secure communication in Chatter application. Unlike classical cryptography, modern cryptography is based on known mathematical problems which means that even if the attacker knows the algorithm/code it could not compromise the information (Table 1). The only way attackers can compromise the system is by knowing the private key, unlike for example Atbash cipher where knowing the cipher algorithm can compromise the data. With these characteristics, Chatter application can be an open-source application and everyone can check the implemented security mechanism.

Classic Cryptography	**Modern Cryptography**
Manipulates traditional characters (letters and digits directly)	Operates on binary bit sequences.
Mainly based on “security through obscurity”. The techniques employed for coding were kept secret and only the parties involved in communication knew about them	Relies on publicly known mathematical algorithms for coding the information. Secrecy is obtained through a secrete key which is used as the seed for the algorithms. The computational difficulty of algorithms, absence of the secret key, etc., make it impossible for an attacker to obtain the original information even if he knows the algorithm used for coding
Requires the entire cryptosystem for communicating confidentially.	Modern cryptography requires parties interested in secure communication to possess the secret key only
Manipulates traditional characters (letters and digits directly)	Operates on binary bit sequences.

Table 2. Comparison between classic and modern cryptography

In modern cryptography, the encryption of plain text can be done in one of two ways. The first way is a stream cipher, and the second way is a block cipher. In stream cipher, every bit in the plaintext will combine with pseudorandom key as a way to encrypt the plaintext bit by bit. In block cipher, many bits in the plaintext will be combined to be processed with a single key to produce the same size of block as a ciphertext [6]. Key distribution can be symmetric (same key for encryption and decryption) or asymmetric (different keys for encryption and decryption).

3.1. Symmetric algorithms

Symmetric key cryptography means that the same key is used to encrypt the message and to decrypt it, making the keys "symmetric". Symmetric algorithms are always faster than asymmetric but do have the issue of exchanging the keys. Symmetric cipher can be easily done using exclusive OR (XOR) function which encrypts text as a stream cipher. XOR is a logical operation that outputs true only when inputs differ (Table 2).

Input		**Output**
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

Table 3. The truth table for XOR operation

The most important feature of XOR operation is its reversibility. If plain text is "01101" and the private key is "01010" then XOR operation will produce ciphertext: "00111". If ciphertext "00111" is again put through XOR operation output will be the same plain text as in beginning "01101". More complex symmetric algorithms are block ciphers that use blocks of data. The most popular algorithms in that area are DES and AES. AES is an asymmetric algorithm based on a design principle known as a substitution-permutation network and is efficient in both software and hardware [7]. It has a fixed block size of 128 bits and a key size of 128, 192, or 256 bits. AES operates on a 4x4 matrix called state. AES consists of four rounds which have several processing steps:

1. Key expansion – round keys are derived from the cipher key using key schedule
2. Initial round:
    a. Add round key – each byte of the state is combined with the round key using bitwise XOR
3. Rounds:
    a. Sub bytes – a non-linear substitution step where each byte is replaced with another according to a lookup table
    b. Shift rows – a transportation step where each row of the state is shifted cyclically a certain number of steps
    c. Mix columns – a mixing operation that operates on the column of the state, combining the four bytes in each column
    d. Add round key
4. Final round:
    a. Sub bytes
    b. Shift rows
    c. Add round key

Even though symmetric algorithms like AES are currently impossible to break, their usage in chatting applications is problematic because sending the key over the network can be insecure (intercepted) and also usage of the application can get more complicated. If the application requires that the user exchanges a key in some other way (example: by SMS), usage of it gets complicated and the user will probably use short passwords which are prone to dictionary attacks.

3.2. Asymmetric algorithms

Asymmetric key cryptography means that there are two keys, one for encryption, one for decryption. One of the most popular asymmetric key algorithms is RSA. Using two separate keys in cryptography gives us more flexibility in exchanging keys. One of the most popular protocols for exchanging keys over an insecure channel is Diffie-Hellman algorithm. Diffie-Hellman key exchange establishes a shared secret between two parties that can be used for secret communication for exchanging data over a public network.

For a better understanding of the concept, we usually use Alice and Bob analogy. Alice and Bob want to exchange a secret key over an insecure network:

1. Alice and Bob publicly agree to use a modulus p and base q ($0 < q < p, n = g^k \ mod \ p$)
2. Alice chooses secret integer a, then sends Bob $A = \ g^a \ mod \ p$
3. Bob chooses a secret integer b, then sends Alice $B = \ g^b \ mod \ p$
4. Alice computes $s = \ B^a \ mod \ p$
5. Bob computes $s = \ A^b \ mod \ p$
6. s from Alice computation and s from Bob computation are equal s=s, which means they successfully exchange data s

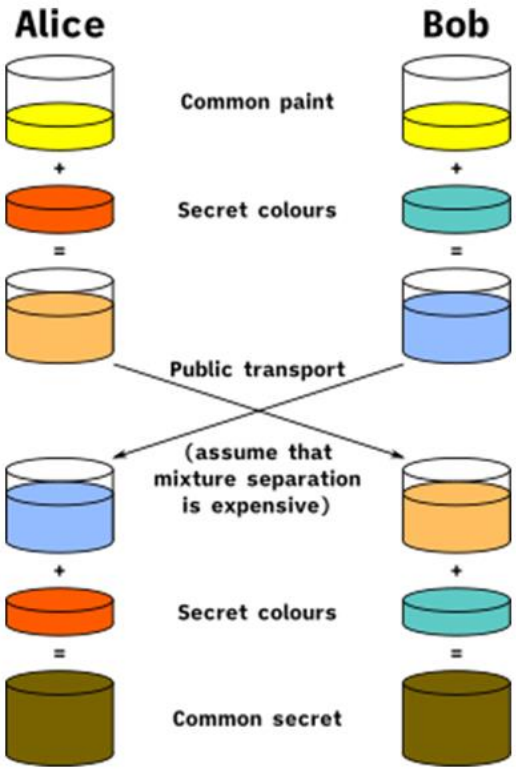

Fig. 4. Alice and Bob communicate by using Diffie-Hellman algorithm (https://www.comparitech.com/blog/information-security/diffie-hellman-key-exchange/)

The algorithm of Alice and Bob communication can be shown by using colors as analogies (Fig. 4). The problem with using Diffie-Hellman protocol is the man-in-the-middle attack. An attacker could intercept the channel and behave as a middle point – communicate with Alice with some private-public pair and communicate with Bob with other private-public key pair. Alice does not know about Bob, and Bob does not know about Alice, so an attacker could easily become a middle point.

Man-in-the-middle attacks could be avoided using public key infrastructure (PKI). The PKI is a set of roles, policies, hardware, software, and procedures needed to create, manage, distribute, use, store and revoke digital certificates and manage public-key encryption. PKI can be viewed as a contract that connects the public key with a specific user and that contract is defined in central authority (CA). Modern protocols use this technique to identify users. This leads us to the next requirement for our Chatter application - it should use HTTPS for communication.

The most famous of the public key cryptosystem is RSA which is named after its three developers Ron Rivest, Adi Shamir, and Leonard Adleman. At the time of the algorithm's development (1977), the three were researchers at the MIT Laboratory for Computer Science [8]. Security of the RSA lies in multiplying two prime numbers (ex. 15 = 3 x 5). Multiplying two or more numbers (factors) is an easy task but finding factors of one number is complicated - especially if we're discussing two large prime numbers.

The RSA involves four steps: key generation, key distribution, encryption and decryption. The keys for the RSA algorithm are generated in the following way:

1. Choose two distinct prime numbers p and q
2. Compute $n = pq$
3. Compute $m = (p-1)(q-1)$
4. Choose a small number e that doesn't have the same factors
   Find d for which this equation is true: $de \,\%\, m = 1$

Public keys are n and e, and private key is d. Numbers p, q and m can all be discarded after d is computed. Chatter application stores public and private key (generated on user login) on local storage:

```
generateKeyPair() {
   var keyPair = keypair();
   localStorage.setItem("pk", keyPair.private);
   return keyPair.public;
 }
```

Message is encrypted with public key and decrypted with private key, which is done like this:

```
encryptMessage(message: string, publickey: string) {
  var publicKey = forge.pki.publicKeyFromPem(atob(publickey));
  var encrypted = publicKey.encrypt(forge.util.encodeUtf8(message));
  return encrypted;
 }

 decryptMessage(message: string) {
  var privatekey = forge.pki.privateKeyFromPem(localStorage.getItem("pk"));
  var decrypted=privatekey.decrypt(message);
  return forge.util.decodeUtf8(decrypted);
}
```

Communication can be verified using hash functions. A hash function is a function that takes a variable-size input and returns a fixed-size string. A hash function is easy to compute, one-way, and collision-free. Hash functions are hard to invert which means that it's computationally not feasible to find the original text from the hash value. To make dictionary attacks more difficult with hash functions, we usually use salting. Salt is a random set of bits that are used as one of the inputs to the hash. The most widely used hash algorithm today is SHA, specifically the SHA-2 algorithm. SHA-2 consists of two similar hash functions - SHA-256 and SHA-512 - that have different block sizes, 256 and 512 bits.

When padding is important, mask generation functions are used. Padding is adding nonsense data to the beginning, middle, or end of a message before encryption. In that way, the same plaintext can have multiple different ciphertexts. A mask generation function (MGF) is a cryptographic primitive similar to a cryptographic hash function except that while a hash function's output is a fixed size, an MGF supports variable-length output. Chatter application is using RSASSA-PSS signatures (Fig. 5) for verifying messages. Probabilistic Signature Scheme (PSS) is a cryptographic signature scheme, and in Chatter application it’s used as SHA 512 hash function with MGF1 masking function and a 20-byte salt. Code for verification and generating signature is:

```
createSignature(message) {
  var md = forge.md.sha512.create();
  var pss = forge.pss.create({
   md: forge.md.sha512.create(),
   mgf: forge.mgf.mgf1.create(forge.md.sha512.create()),
   saltLength: 20
  });
  var privatekey = forge.pki.privateKeyFromPem(localStorage.getItem("pk"));
  md.update(message,'utf-8');
  return privatekey.sign(md,pss);
 }

 verifySignature(signature,message,publickey){
  publickey=atob(publickey);
  var publicKey=forge.pki.publicKeyFromPem(publickey);
  var md = forge.md.sha512.create();
  var pss = forge.pss.create({
   md: forge.md.sha512.create(),
   mgf: forge.mgf.mgf1.create(forge.md.sha512.create()),
   saltLength: 20
  });
  md.update(message,'utf-8');
  return publicKey.verify(md.digest().getBytes(),signature,pss);
 }
```

There are other methods that we could use to further improve the security of our application. In the future, when available, we could look into using quantum cryptography, as it’s the logical next step to be used in secure communication. Chatter application could use this idea as well, to further increase the security level as end users want to have as much privacy as possible, and secure communication is one of the key parameters in that process. After all, it is necessary to strike a balance between the freedoms of individuals on the Internet; their rights should not be violated [9]. Not all methods need to be just encryption-based, they could also be context-based. We could look into neural networks to classify individual data packets so that we can implement some sort of classification-based mechanism to reject packages aimed at our Chatter application that are coming from unknown or insecure sources - for example, sources that are not involved in the chat session[10].

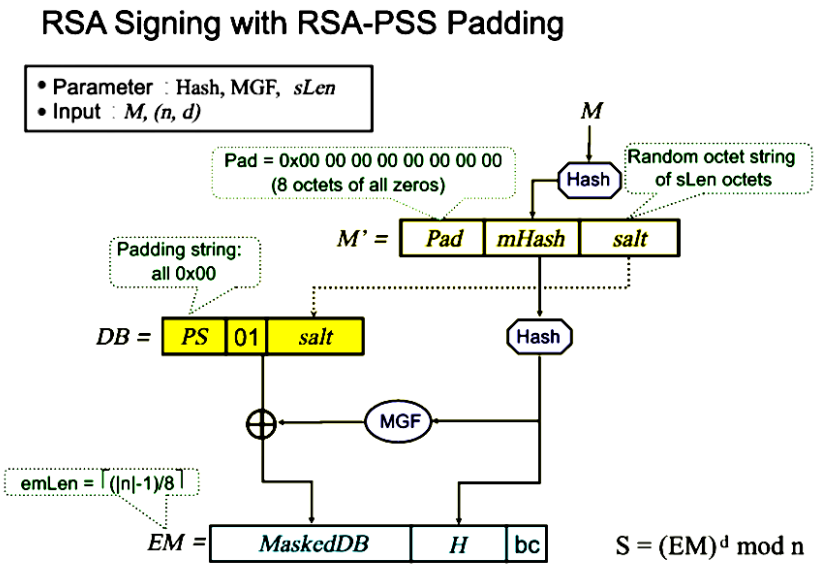

Fig. 5. Diagram of RSA signing with RSA-PSS padding
(https://slidetodoc.com/presentation_image/a6b6804dc2d0898c8a752ad3552d4ea3/image-53.jpg)

This would also fit well with the idea of layered security, where we don't use just one technology or component to deliver a secure solution - we use multiple different solutions on different levels. We also see these two concepts as potential areas for future study and improvement for our application.

4. Conclusion

Implementing a secure chatting application on modern computers is impossible with traditional cryptography. Modern cryptographic algorithms can be divided into symmetric and asymmetric ciphers. Symmetric ciphers are faster to compute but have a problem in exchanging keys over insecure networks. Communication can be verified by using hash functions with extra padding. This also adds extra obfuscation to the data, so it makes breaking it even harder. By using the session approach, each user gets a new key with a new login, older communications could not be compromised. Our research indicates that, by using a hybrid approach, the asymmetric algorithm could be used to transfer symmetric keys over an insecure network, and then use symmetric cipher for securing the data. A popular algorithm for implementing this hybrid approach is RSA, which is used in our Chatter application. We find this methodology to be the best in terms of the price/performance ratio of computational power used vs time-based overhead that gets introduced into the communication process.

5. References

[1] Roser, M., Ritchie, H. & Ortiz-Ospina, E. (2015). Our world in data, Available from: https://ourworldindata.org/internet#growth-of-the-internet, Accessed: 2021-01-31
[2] European Parliament and Council of European Union (2016). Regulation (EU) 2016/679. Available at: https://gdpr-info.eu/art-4-gdpr/, Accessed: 2021-01-31
[3] Aziz, B. & Hamilton, g. (2009). Detecting man-in-the-Middle Attacks by Precise Timing, Third International Conference on Emerging Security Information, Systems and Technologies
[4] Cohen, F. (1987). Introductory Information Protection, ASIN: B000715AKW
[5] Jonathan K. & Yehuda L. (2014). Introduction to Modern Cryptography, Chapman and Hall/CRC, 978-1466570269, New York
[6] Bokhari, M. U. & Shallal, Q. M. (2016). A Review on Symmetric Key Encryption Techniques in Cryptography, International Journal of Computer Applications, Volume 147 - Number 10, 0975 – 8887
[7] Bruce S. et al. (2000). The Twofish Team's Final Comments on AES Selection, unpublished
[8] Christensen, C. (2006), Introduction to RSA and to Authentication, unpublished
[9] Zharova, A., Elin, V. & Panfilov, P. (2018). Technological and legal issues of identifying a person on the internet to ensure information security, Annals of DAAAM and Proceedings of the International DAAAM Symposium, 29(1), pp. 0471–0478. doi: 10.2507/29th.daaam.proceedings.069.
[10] Igor, H. et al. (2014). Application of neural networks in computer security, 24th DAAAM International Symposium on Intelligent Manufacturing and Automation, 2013. doi: 10.1016/j.proeng.2014.03.111.