Pontifícia Universidade Católica do Rio de Janeiro

Marcio Ricardo Rosemberg

HX: A Proposal of a New Stream Cipher Based on Collision Resistant Hash Functions.

Tese de Doutorado

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências - Informática.

Advisor: Prof. Marcus Vinícius Soledade Poggi de Aragão

Rio de Janeiro

April 2019

Pontifícia Universidade Católica do Rio de Janeiro

Marcio Ricardo Rosemberg

HX: A Proposal of a New Stream Cipher Based on Collision Resistant Hash Functions.

Thesis presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Doutor em Ciências - Informática. Approved by the undersigned Examination Committee.

Rio de Janeiro, April 26th, 2019

Marcio Ricardo Rosemberg

M.Sc. Informatics (PUC-Rio – 2014). BSc. Electrical Engineering (Universidade Santa Úrsula – 1990). Managing partner and founder of SYSNET Sistemas e Redes – 1994-2019, a company that specializes in networking, network security, and development of customized software. The author has been asked several times to provide consulting as an expert for judges on several lawsuits.

Rosemberg, Marcio Ricardo

HX: A Proposal of a New Stream Cipher Based on Collision Resistant Hash Functions. / Marcio Ricardo Rosemberg; advisor: Marcus Vinicius Soledade Poggi de Aragão – 2019.
173f : il. (color.) ; 30 cm

Tese (doutorado) – Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2019.
Inclui bibliografia

1. Informática – Teses. 2. Criptografia. 3. Autenticação. 4. Cifras de Fluxo. 5. Negociação de Chaves Criptográficas. I. Aragão, Marcus Vinicius Soledade Poggi de. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. IV. Título.

CDD: 004

This work is dedicated to my grandmother, Fany Bass Rosemberg Z’L.

Acknowledgments

To CNPq and PUC-Rio, for the scholarship, which made possible the realization of this work.

To my advisor, Professor Marcus Poggi for his support, patience and teachings.

To Professor Daniel Schwabe for his teachings and encouragement.

To the other professors of the Departamento de Informática of PUC-Rio.

To my wife Adriana for her love and incentive.

To my daughter Nicole.

To my family.

To all my friends in PUC-Rio

To all other people who helped me directly or indirectly

Abstract

Rosemberg, Marcio Ricadro; de Aragão , Marcus Vinicius Soledade Poggi (advisor); **HX: A Proposal of a New Stream Cipher Based on Collision Resistant Hash Functions.** Rio de Janeiro, 2019. 173p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

In the near future, we will live in smart cities. Our house, our car and most of our appliances will be interconnected. If the infrastructure of the smart cities fails to provide privacy and security, citizens will be reluctant to participate and the main advantages of a smart city will dissolve. Several encryption algorithms have been broken recently or significantly weakened and key lengths are increasing as computing power availability grows. In addition to the ever growing computing power a recent study discovered that 93% from 20,000 Android applications had violated one or more cryptographic rules. Those violations either weaken the encryption or render them useless. Another problem is authentication. A single compromised private key from any intermediate certificate authority can compromise every smart city which will use digital certificates for authentication. In this work, we investigate why such violations occur and we propose: HX, a modular encryption algorithm based on Collision Resistant Hash Functions that automatically mitigates cryptographic rules violations and HXAuth, a symmetric key authentication protocol to work in tandem with Secure RDF Authentication Protocol (SRAP) or independently with a pre-shared secret. Our experiments points in the direction that most developers do not have the necessary background in cryptography to correctly use encryption algorithms, even those who believed they had. Our experiments also prove HX is safe, modular and is stronger, more effective and more efficient than AES, Salsa20 and HC-256.

Keywords

Stream Ciphers; Cryptography; Authentication; Hash; Key Management.

Resumo

Rosemberg, Marcio Ricardo; de Aragão, Marcus Vinicius Soledade. **HX: Uma Proposta de Uma Nova Cifra de Fluxo Baseada em Funções de Hash Resistentes à Colisão.** Rio de Janeiro, 2019. 173p. Tese de Doutorado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

No futuro próximo, viveremos em cidades inteligentes. Nossas casas, nossos carros e a maioria dos nossos equipamentos estarão interconectados. Se a infraestrutura das cidades inteligentes não fornecerem privacidade e segurança, os cidadãos ficarão relutantes em participar e as principais vantagens de uma cidade inteligente irão se dissolver. Vários algoritmos de criptografia recentemente foram quebrados ou enfraquecidos e os comprimentos das chaves estão aumentando, conforme cresce o poder computacional. Um estudo recente descobriu que 93% de 20.000 aplicações Android tinham violado uma ou mais regras de criptografia. Essas violações enfraquecem a criptografia ou as inutiliza. Outro problema é a autenticação. Uma chave privada comprometida de única autoridade de certificação intermediária pode comprometer toda cidade inteligente que utilizar certificados digitais para autenticação. Neste trabalho, investigamos por que tais violações ocorrem. Propomos o HX: um algoritmo de criptografia modular baseado em funções de hash resistentes à colisão que reduz automaticamente as violações de regras de criptografia e o HXAuth: um protocolo de autenticação de chave simétrica para trabalhar em conjunto com o SRAP ou independentemente, com um segredo previamente partilhado. Nossos experimentos apontam na direção de que a maioria dos desenvolvedores não tem o conhecimento básico necessário em criptografia para utilizar corretamente um algoritmo de criptografia. Nossos experimentos também provam que o HX é seguro, modular e é mais forte, mais eficaz e mais eficiente do que o AES, o Salsa20 e o HC-256.

Palavras-chave

Cifras de Fluxo; Criptografia; Autenticação; Resumos Criptográficos; Gerência de Chaves.

Contents

List of Figures

List of Tables

Acronyms

3DES	Triple DES
AES	Advanced Encryption Standard
AMD	Advanced Micro Devices
AP	Authentication Partner
API	Application Programming Interface
APLR	Authentication Partner of Last Resort
ARX	Add-Rotate-Xor
CA	Certification Authorities
CBC	Cipher Block Chaining
CFB	Cipher Feedback Mode
CHAP	Challenge Handshake Authentication Protocol
CPU	Central Processing Unit
CRHF	Collision Resistant Hash Function
CRYPTREC	Cryptography Research and Evaluation Committees
CSV	Comma Separated Values
CTR	Counter Mode
CWI	Centrum Wiskunde & Informatica
DDR	Double Data Rate
DES	Data Encryption Standard
DG	Default Gateway
DoS	Denial of Service
ECB	Electronic Code Block
ESP	Encapsulated Security Payload
FEAL	Fast data Encipherment Algorithm
ESP	Encapsulated Security Payload
GB	Gigabyte
GPS	Global Positioning System
GPU	Graphics Processing Unit
H0	Null Hypothesis
HA	Alternative Hypothesis
HMAC	Hash Based Message Authentication Code
HX	Hash and XOR
IBM	International Business Machines
ICM	Integer Counter Mode
ID	Identity
IDEA	International Data Encryption Algorithm
IEC	International Electrotechnical Commission
IP	Internet Protocol
ISO	International Organization for Standardization
IV	Initialization Vector or Initial Value
KB	Kilobyte
LAN	Local Area Network
MAC	Message Authentication Code
MB	Megabyte

Mbps	Magabits per second
MMS	Multimedia Messaging Service
NAT	Network Address Translation
NESSIE	New European Schemes for Signatures, Integrity and Encryption
NIC	Network Interface Card
NIST	National Institute of Standards and Technology
Nonce	Number Only Once
NSA	National Security Agency
NTP	Network Time Protocol
OFB	Output Feedback
OS	Operating System
OWL	Web Ontology Language
PC	Personal Computer
PIN	Personal Identification Numbers
PKI	Public Key Infrastructure
PRF	Pseudorandom Function
PRP	Pseudorandom Permutation
RACE	Research and Development in Advanced Communications Technologies in Europe
RAM	Random Access Memory
RC4	Ron's Code 4 or Rivest Cipher 4
RDF	Resource Description Framework
RDFK	Hidden RDF
RIPEMD	RACE Integrity Primitives Evaluation Message Digest
RQ	Research Question
SHA	Secure Hash Algorithm
SIC	Segmented Integer Counter
SMS	Short Message Service
SRAP	Secure RDF Authentication Protocol
SSD	Solid State Disk
SSL	Secure Sockets Layer
TB	Terabyte
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UBI	Unique Block Iteration
URI	Uniform Resource Identifier
USD	United States Dollar
VCPU	Virtual CPU
VM	Virtual Machine
VoIP	Voice Over IP
VPN	Virtual Private Network
WAN	Wide Area Network
WEP	Wired Equivalent Privacy
Wi-Fi	Wireless Fidelity or Wireless Networking
WPA-TKIP	Wi-Fi Protected Access - Temporal Key Integrity Protocol
XOR	Exclusive Or

1 Introduction

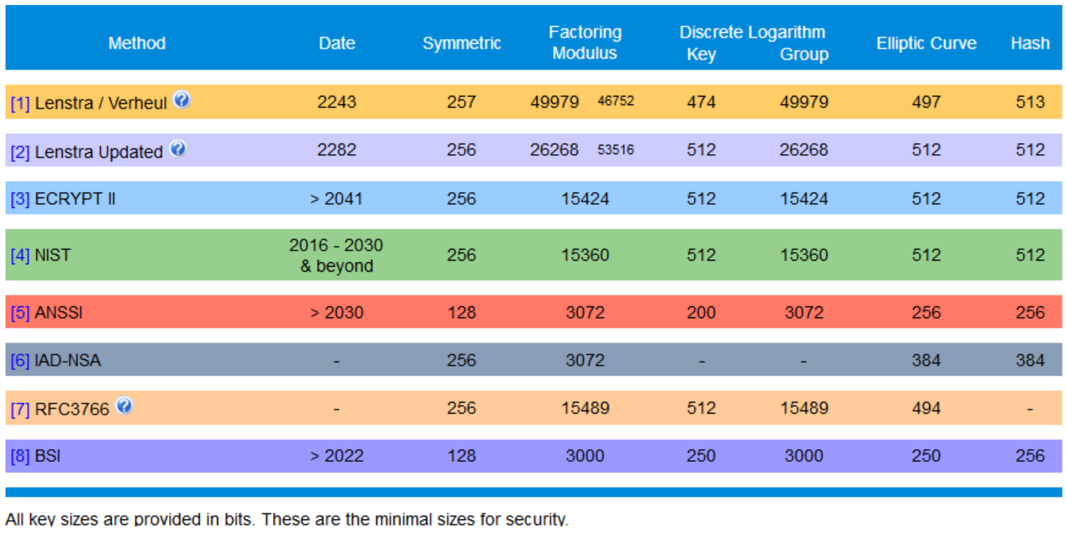
In this chapter, we discuss the challenges of privacy and security for smart cities in the near future. We present the motivation of our research, its main contributions and how the thesis is organized.

1.1 The Problem

In the near future, we will live in smart cities. Our houses, our cars and most of our appliances will be interconnected. Our mobile devices will collect huge amounts of data and will send them to clouds for real time processing (McLaren, D; Agyeman, J, 2015) (Musa, 2016). Meanwhile, smart cities must ensure individual privacy and security. If the infrastructure fails to provide privacy and security, citizens will be reluctant to participate and the main advantages of a smart city will dissolve (Braun, et al., 2018).

Currently the e-commerce share of the total global retail market is about 12%, accounting for USD 161 billion at the end of 2016. It is projected to jump to USD 319 billion by 2020 (about 17.5%) (Lui, 2018). All of those transactions must be secure. Users must be correctly authenticated and the communications between endpoints must be encrypted, in order ensure confidentiality. However, several encryption algorithms have been recently weakened or broken. The use of a compromised encryption algorithm impacts on the confidentiality. As a result, new and stronger cryptographic algorithms, with larger key sizes, are needed to replace obsolete or weakened cryptographic algorithms.

The near future will require symmetric encryption keys with a minimum length of 256 bits. The NSA (National Security Agency) is already recommending 256 bit key lengths for any material classified up to TOP-SECRET (NSA - National Security Agency, 2016).



Method	Date	Symmetric	Factoring Modulus	Discrete Logarithm		Elliptic Curve	Hash
				Key	Group		
[1] Lenstra / Verheul	2243	257	49979 46752	474	49979	497	513
[2] Lenstra Updated	2282	256	26268 53516	512	26268	512	512
[3] ECRYPT II	> 2041	256	15424	512	15424	512	512
[4] NIST	2016 - 2030 & beyond	256	15360	512	15360	512	512
[5] ANSSI	> 2030	128	3072	200	3072	256	256
[6] IAD-NSA	-	256	3072	-	-	384	384
[7] RFC3766	-	256	15489	512	15489	494	-
[8] BSI	> 2022	128	3000	250	3000	250	256

All key sizes are provided in bits. These are the minimal sizes for security.

Figure 1 - Key Length Recommendations for the Near Future (Giry & Quisquater, 2017)

Another problem, from the developer's point of view, is that cryptographic algorithms are not trivial to use or parameterize. Cryptographic algorithms are based on complex mathematical equations. However, parameterizing such equations in a friendly interface is not an easy task.

According to the 6th principle of Kerckhoff [2.1], a cipher system must be easy to use. Unfortunately, less importance has been given to this principle. Most encryption algorithms have a considerable list of arguments or attributes. If those attributes are not set correctly, the encryption method may either fail when invoked or may produce a weak ciphertext. In order to avoid such mistakes, a set of common rules or best-practices were compiled to support developers in the correct use of cryptographic algorithms. The NIST (National Institute of Standards and Technology) has several publications available for download about key management, security and privacy control for organizations and others (Barker, 2016). Developers do not need to implement the encryption algorithms. They are available in most programming languages. Nevertheless, developers must correctly use the encryption API by not making mistakes that may compromise the encryption effectiveness.

Several tools have been proposed to detect violations of the best-practice rules in privacy and security. In a case study published by Egele *et al.* (Egele, et al., 2013), the authors developed a tool to detect cryptographic rules violated by developers on Android applications. In their work, they chose six common rules in cryptography and verified how many Android applications violated at least one of those rules and how many had two distinct violations or more. Their study revealed that, from almost 20,000 applications, only 7% of the applications have not violated any of the rules. Nevertheless, the vast majority (93%) had. Also, 30% of the applications violated two rules.

Violations related to weak encryption keys, weak encryption algorithms and weak random generators were studied by Balebako *et al*. (Balebako, 2014). Although the authors found problems in some cryptographic API implementations, the vast majority of problems were related to misuse of the cryptographic APIs. The authors concluded “there was a lack of understanding around privacy best-practices” among the participants they interviewed. Those conclusions were reinforced by Lazar *et al*. (Lazar, 2014). They analyzed mistakes in systems that implemented and used cryptographic algorithms and demonstrated that 83% of the bugs related to privacy and security are related to misuse of the cryptographic libraries, while 17% are bugs in the cryptographic libraries themselves.

1.2 Motivation

In our Master’s Dissertation, we proposed SRAP (Secure RDF Authentication Protocol) (ROSEMBERG, 2014). SRAP is a mutual authentication and key exchange protocol, designed for Semantic Web applications. SRAP decentralizes the trusted third party, making use of certificate authorities only as a last resort, when the client does not trust any of the authentication partners. SRAP also changes the authentication paradigm by caching URIs and public keys from previous authenticated servers or clients into a social graph, unlike TLS (Transport Layer Security), which verifies certificates in every authentication. In SRAP, when the public keys have already been authenticated, both endpoints can establish direct communication, verifying only the proof of possession of each corresponding private key.

During the presentation, the review board suggested, since SRAP changes the authentication paradigm in the second and subsequent authentications, such authentications could be achieved with a symmetric key protocol, resulting in a much faster authentication process.

Through the course of our doctorate studies, we found out several symmetric key encryption algorithms have been broken or significantly weakened. RC4 in WEP (Wired Equivalent Privacy) was broken in 2007 (Tews, et al., 2007) and in TLS or WPA-TKIP in 2016. In 2016 (Vanhoef & Piessens, 2015), Triple DES and Blowfish were also proven to be unsecure. Both Blowfish and Triple DES are vulnerable to SWEET32 attacks (Bhargavan & Leurent, 2016). Even in AES (Advanced Encryption Standard), which is considered the state of the art in symmetric key encryption, bugs were found in its key scheduling algorithm, resulting in the decrease of its complexity (Biryukov & Khovratovich, 2009) (Bogdanov, et al., 2011).

In many APIs such as Java Cipher Class, Python Crypto Package or .NET Framework Cryptography Class, from the current set of available encryption algorithms, only AES is still safe to use and widely accepted (National Security Agancy/Central Security Service/Information Assurence Directorate, 2016). There are other safe encryption algorithms like Twofish, HC-256 or Salsa20/ChaCha, but they are usually not native in most common APIs. Their use will require either implementation by the developers or copying some source code of those algorithms from forums or other resources, which may not be safe.

Computing power is constantly growing and attacks on keys are becoming more sophisticated. As a result, the easier the attacks on keys, the larger key lengths become necessary.

Taking into account all these factors and particularly the mistakes developers make when they use encryption algorithms, we were motivated to research and specify a symmetric encryption algorithm, named HX (Hash and XOR), based on Collision Resistant Hash Functions (CRHF) with the following requirements:

- Requires only the plaintext and the encryption key for encryption or ciphered text and the encryption for decryption as mandatory parameters.
- Developers optionally can specify a hash algorithm and a cipher mode
- Keys can be of any length
- No padding scheme is needed
- No real time synchronization required between the sender and the receiver
- Easy to implement on almost any platform and programming language
- Automatic handling of all random parameters
- User friendly interface

- Flexibility and Modularity

We define a user Friendly Interface as a simple set of methods and attributes, reasonably well documented able to automatically mitigate several encryption rules violations.

We define Flexibility and Modularity as the capability for the user or software developer to choose a CRHF from a set, according to her security requirements. If the CRHF becomes unsecure in the future, all she has to do is to adopt another CRHF. Neither the encryption nor the decryption algorithm needs changes in their coding.

Future versions of the HX class or package may incorporate new and stronger CRHF. However, from the developer's point of view, there will be no change in the code they produce other than setting the new algorithms for encryption or decryption.

1.3 Goals and Contributions

- The specification of a symmetric key authentication and session key negotiation protocol for SRAP second time and subsequent authentications. The use of a symmetric key authentication protocol improves the authentication speed and security over SRAP's Fast Negotiation (ROSEMBERG, 2014) pp. 65.
- The design and implementation of a symmetric key encryption algorithm capable of accepting encryption keys of 160, 256, 384 and 512 bits or more. A 384 or higher key length is a significant improvement over current 256 bits length algorithms.
- An empirical study on best practice rules violations commonly made by developers, when using symmetric key cryptography algorithms and a usability comparison with the proposed algorithm. This study also demonstrates the immaturity of developers in relation to basic cryptography concepts necessary to protect privacy and security.

1.4 Research Questions

RQ$_1$: Are software developers familiarized with the basic concepts of symmetric key cryptography?

Egele *et al.* (Egele, et al., 2013), Balebako *et al.* (Balebako, 2014) and Lazar *et al.* (Lazar, 2014) studied cryptographic rules violations. They did not, however, identified the root cause of the problem. Do software developers even know what an encryption key is? Do they understand the parameters they need to pass to encryption functions? If they do not, this issue can be mitigated by reducing the parameters list and educating them on key management, which is exactly what Kerckhoff proposed [2.1].

RQ$_2$: Are developers capable of consciously choosing a safe symmetric key encryption algorithm?

This is more difficult for the developers. Unless they keep track on cryptanalysis news, it would be difficult for them to select a safe algorithm and to be sure they made a safe choice. If they choose a block cipher algorithm, they must also select a safe mode of operation [2.4.3]. Such skills have to be verified. Nevertheless, if a cryptosystem is “modular”, meaning it has a uniform high level set of attributes and methods, software developer languages updates can deprecate low level algorithms (algorithms used for Pseudorandom Permutation – PRP or Pseudorandom Function – PRF). Advising deprecated algorithms should be used only in legacy applications. Those updates can also change the default low level algorithm. The term “safe” in this thesis means the algorithm cipher is secure and therefore, reliable. The algorithm is neither broken nor significantly weakened. [2.1].

RQ$_3$: Can developers use the AES (CTR Mode) encryption algorithm correctly?

The AES-CTR is one of the fastest and safest encryption algorithms and modes of operation [2.4.3], [2.4.4.7]. In addition, AES is likely to be the only safe algorithm commonly available in most APIs. Nevertheless, it is extremely sensitive to its parameters list. If not used correctly, confidentiality may not be achieved. We intend to confirm if developers can use AES-CTR correctly.

RQ_4: is it possible to use a CRHF in combination with a secret key to generate one time pads that will not repeat itself for a long time?

If a CRHF in combination with a secret key can be used as a PRF. It could also be used as a keystream generator. As a result, we could build a stream cipher algorithm from a CRHF by applying a bitwise XOR operation between the keystream and the plaintext for encryption and a bitwise operation between the keystream and the ciphered text for decryption.

RQ_5: How effective is HX when compared to other encryption algorithms?

If the null hypothesis is rejected for RQ_4, HX and other encryption algorithms have to be compared for effectiveness and efficiency. Effectiveness, in the usability context, means less violation of the following cryptographic rules.

1. Do not use the ECB mode of operation if the message length is bigger than algorithm's block length.
2. Always use a random Initialization Vector
3. Do not use a weak encryption key
4. Do not use a broken or weakened encryption algorithm

Those rules were selected taking into account that AES [2.4.4.7] and Twofish [2.4.4.5] are the two most commonly available encryption algorithms that are safe to be used. Both require an Initialization Vector [2.2] and a mode of operation [2.4.3] among other parameters. Nevertheless, the developer may choose an unsafe encryption algorithm, such as DES [2.4.4.1], 3DES [2.4.4.2], Blowfish [2.4.4.4] or RC4 [2.4.4.8].

The effectiveness test, assumes HX is safe to use. However, at this time, an assertion such as this cannot be considered definitive, for the reason that an encryption algorithm takes years to be certified (Preneel, 2007).

RQ_6: How efficient is HX when compared to other encryption algorithms?

We can perform an efficiency test, where efficiency, in the usability context, means the time spent to implement a code that encrypts and decrypts messages with HX or other algorithms.

1.5 Related Work

Bruce Schneier proposed a simple way to use One-way Hash Functions to encrypt data in a stream cipher algorithm (Schneier, 1996):

$$C_i = M_i \oplus H(K \parallel C_{i-1}) \text{ and } M_i = C_i \oplus H(K \parallel C_{i-1})$$

This is basically the use of a Hash algorithm in Cipher Feedback Mode (CFB) [2.4.3] to generate one-time pads. Schneier said nothing about C_0. C_0 can be H(K) or it could be significant strengthened by using an Initialization Vector H(K,IV). In the previously mentioned formula $\oplus$ means the bitwise XOR operation and $\parallel$ means string concatenation.

Another encryption algorithm, using Hash algorithms to use one time pads was published in 1999 by Peyravian, *et al.* (Peyravian, et al., 1999). In this cryptosystem, the authors used the SHA-1[2.5.3] hash of the encryption key and a sequential counter to generate a keystream. They called this cryptosystem: Hash-based Encryption System.

The encryption/decryption algorithm is described as:

$$C_i = M_i \oplus H(I \parallel K) \text{ and } M_i = C_i \oplus H(I \parallel K).$$

The ciphered text is $C = C_1 \parallel C_2 \parallel \ldots \parallel C_n.$

This hash based stream cipher was cited 18 times. The articles (Gordon & Loeb, 2002), (Campbell, et al., 2003), (Lee, et al., 2006), (Wang, et al., 2011), (Kumari, et al., 2012) (Kumari & Khan, 2014), (Gordon & Loeb, 2001), (Gordon & Loeb, 2004), (Demirkan & Goul, 2013), (Patrick, 2008), (Tesink, et al., 2005), (Elzouka, 2006), (Yeh & Chou, 2001), (Elzouka, 2008), (Chen, et al., 2013), (Li, et al., 2003) and (Cheng, 2005) cite this work as background reference. Huang, Feng, & Zhang in 2001 (Huang, et al., 2001), in a four page short paper, propose an encryption scheme based on one-way hash and the services of a pseudorandom number generator to enhance the algorithm.

Specifically for images encryption, Cheddad, et al. in 2010 (Cheddad, et al., 2010) proposed an encryption algorithm that uses hashes and the Fourier Transform.

With both schemes, there are a few problems:

1) Not all hashes can be used in such a way. Most are vulnerable to the Length Extension Attack [2.7.2] and can be exploited with an efficient algorithm.
2) The same plaintext encrypted with the same key will produce the same ciphered text. In Schneier's proposal, this is true if $C_0 = H(K)$.
3) Even if a hash algorithm is resistant to the Length Extension Attack, the scheme proposed by Peyravian *et al.* produces the same keystream when the same key and initial counter is used. Such flaw, leads to a catastrophic security failure. All the attacker needs to do is to capture two ciphertexts encrypted with the same key and initial counter. She can then perform a bitwise XOR between the two ciphertexts to retrieve the keystream. With possession of the keystream, the attacker can decipher all messages encrypted with the same key and initial counter, without the need of the encryption key.

In order to use hash functions to produce keystreams, all these issues have to be dealt with.

1.6 Outline of the Thesis

The thesis is organized in six sections.

In section 2, we provide the necessary background to support our work and conclusions.

In section 3, we describe the research questions, the hypothesis formulated to refute or confirm the research questions, the research methodology and the experiments designs.

In section 4, we state our proposal of the HX encryption algorithm, the HXAuth authentication and key agreement protocol and the security analysis of both the algorithm and the protocol. We also describe the minimum set of attributes and methods necessary for the implementation of HX and HXBlock, a variant of HX designed to encrypt and decrypt streams in real-time applications.

In section 5, we report our experiment results and compare HX with the state of the art of encryption algorithms.

In section 6, we summarize our conclusions and give directions to future works.

In this work, the participants of the experiments as well as the personal pronouns of undetermined genders are always referred as she, regardless of the true gender.

2 Background

In this chapter, we provide the theoretical background necessary to understand this work.

2.1 The Kerckhoff's Principles

In 1883, August Kerckhoff published the principles upon which a cryptographic system should rely (Kerchhoff, 1883).

Ideally Kerckhoff wanted the security of the system to depend only on the choice of the keys. Other rules should be reduced to a minimum set for the success of the system. Translated from French, those principles are:

1) "The system must be practically, if not mathematically, indecipherable".

2) "It should not require secrecy, and it should not be a problem if it falls into enemy hands".

3) "It must be possible to communicate and remember the key without using written notes, and correspondents must be able to change or modify it at will".

4) "It must be applicable to telegraph communications".

5) "It must be portable, and should not require several persons to handle or operate".

6) "Lastly, given the circumstances in which it is to be used, the system must be easy to use and should not be stressful to use or require its users to know and comply with a long list of rules".

Summarizing Kerckhoff's principles, we need a strong public encryption algorithm in such a way that if the attacker gains access to one or more ciphertexts, she must not be able to compute the encryption keys or decipher the ciphertexts. If the attacker is able to obtain both the ciphertexts and the original messages, she must not be able to compute the encryption key and then decipher future messages encrypted with the same key. Finally, the parameters and cryptographic rules the users of the algorithm must comply should be reduced. In this work, we strongly emphasize this principle.

2.2 Salts, Nonces and Initialization Vectors

Salts and Nonces are random data with different goals. Salts protect passwords against dictionary attacks, when passwords are hashed. Combining a password with a salt and then hash the concatenated string, ensures different hashes outputs for the same passwords, as long as the salts are different. In other words, the salt “customizes” the hash of a password of a specific user (Bellare & Rogaway, 2005). Salts are generally used in large quantities. An attacker which gains access to a password database, where the passwords are concatenated with salts and then hashed, cannot simple hash the each password of the dictionary and check if it matches any hash of the database. Instead she must hash each password of the dictionary with every salt in the database and then verify if it matches any stored hash. Hence the number of queries the attacker must perform to find the passwords increases. In other words, the search space is increased.

Nonces (number used only once) are one-time random numbers to be used, for example, as IV (initialization vectors) in encryption algorithms, ensuring that the same plaintext encrypted with the same key produces a different ciphertext. Such technique makes it more difficult for an attacker to find patterns and recover the encryption key or the plaintext (Rogaway, 2002). The initialization vector allows the encryption key to be used longer, avoiding the slow process of rekeying (Huang, et al., 2013). IVs are not secret and should be transmitted openly to the receiver.

Nonces are also used in authentication protocols to mitigate Replay Attacks (Stallings, 2011). Usually there is only one nonce for the entire encryption process of a message, while salts are abundant and preferably unique for each password stored in a repository.

2.3 The Vernam Cipher

The Vernam Cipher or one-time pad is considered the only unbreakable encryption scheme (Kahn, 1967) (Vernam, 1926). It consists of an XOR operation between the plaintext and a random key of the same length of the plaintext. Once encrypted, the ciphertext gives nothing the cryptanalyst can use to decipher the encrypted message (Preneel, 2007).

By studying the Vernam Cipher, Shannon published a mathematical proof, showing that the attacker is unable to obtain any information on the plaintext or original message from the observation of the ciphertext, no matter how much computing power she has. Shannon called this property: “Perfect Secrecy” and he also proved the key of the Vernam Cipher cannot be shorter than the plaintext, if the endpoints want perfect secrecy (Shannon, 1949).

On the other hand, the Vernam Cipher has two drawbacks:

1) It is not feasible to negotiate a key between two parties, in which the key size is as big as the message size, since the secure channel, in which the key must be negotiated must also provide perfect secrecy. If the two parties are able to provide a secure channel capable of negotiating a key with the same length of the message, they could also exchange messages without the need for encryption.

2) For every new message, a fresh random key must be negotiated.

2.4 Symmetric Key Encryption Algorithms

Algorithms that use the same key to encrypt and decrypt messages are called symmetric algorithms. Alice and Bob must agree on a single encryption and decryption key which would be used by both during the session (STALLINGS, 2011).

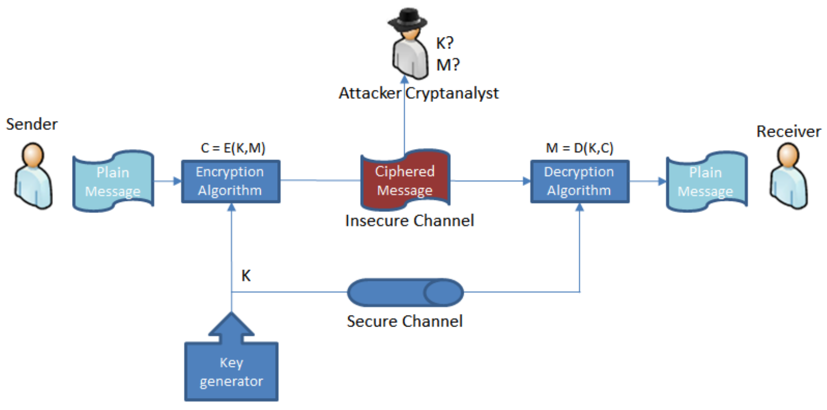


Figure 2 - Model of symmetric key cryptosystem

The problem symmetric encryption algorithms do not solve, is how the endpoints negotiate a session key (a symmetric key used in one communications session), since the encryption algorithm is not responsible for the secure channel in which the session keys are generated or negotiated. Key Management is the area of Cryptography which studies in detail the problem of exchanging keys. Key Management is detailed in [2.9]

Symmetric encryption algorithms are divided in two categories: Stream Cipher and Block Cipher algorithms.

2.4.1 Stream Cipher Encryption Algorithms

The goal of stream ciphers is to mimic the Vernam Cipher by continuously generating and synchronizing new keys between the sender and the receiver (El-Razouk, et al., 2014). They are more difficult to implement than block ciphers, because the keystream cannot repeat itself during the session. There are two types for operation modes: Synchronous Stream Cipher and Self-synchronizing stream ciphers.

In Synchronous Stream Cipher, the keystream is generated independently of the plaintext and of the ciphertext. The keystream is commonly produced by a pseudorandom generator, parameterized by the secret key of the whole scheme. In this mode, the sender and receiver must be synchronized for decryption to be successful. One way to achieve synchronization is to send an Initialization Vector (IV) in the open before each ciphertext (Fontaine, 2011) (Rueppel, 1986).

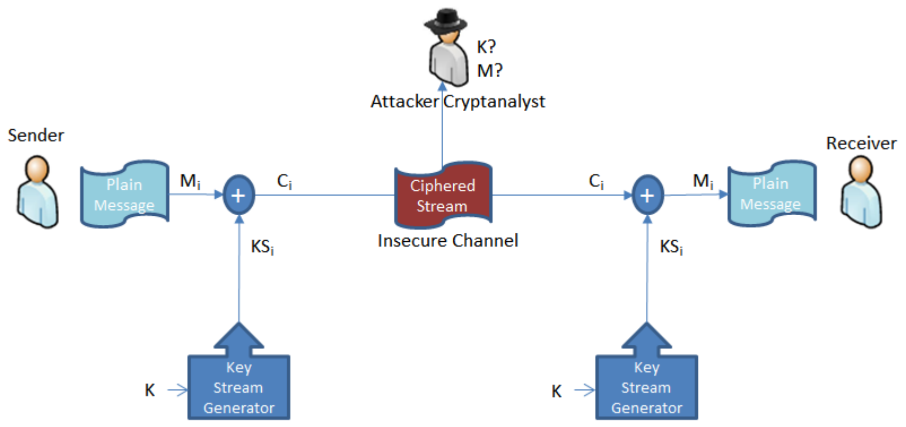


Figure 3 - Synchronous Stream Cipher

In a Self-synchronizing, or asynchronous, stream cipher, the keystream depends on the secret key of the scheme and also on a fixed number of ciphered text digits that have already been produced by the sender, or read by the receiver. The idea of self-synchronization was patented in 1946 and has the advantage that the receiver will automatically synchronize with the keystream generator after receiving a certain number of ciphered text digits (Fontaine, 2011) (Daemen & Kitsos, 2008).

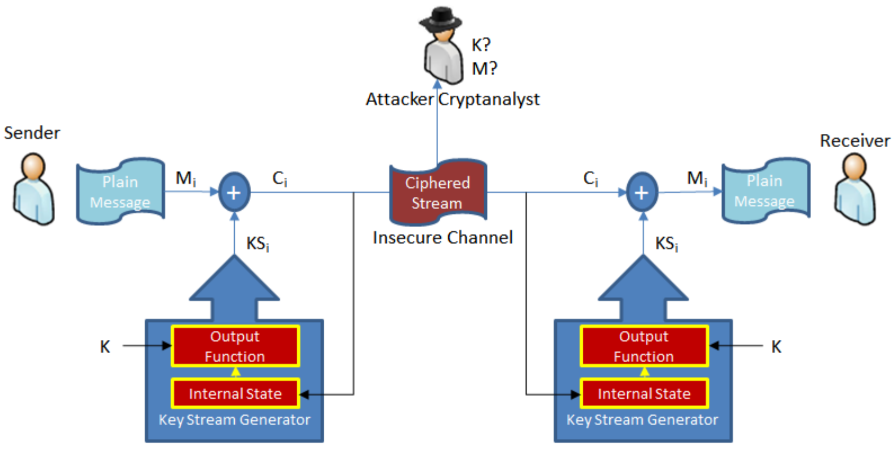


Figure 4 - Self-Synchronizing Stream Cipher

2.4.2 Block Cipher Encryption Algorithms

Block ciphers operate on fixed length blocks of bits (Stallings, 2011). If the last block of the plaintext is shorter than the block size, some sort of padding scheme is required to complete the last block. Block ciphers use the same key on every block to encrypt the plaintext block. They operate on the principle of Pseudorandom Permutation (PRP). Pseudorandom permutation must be invertible or decryption would not be possible. As a result, block ciphers use two different algorithms: one for encryption and another for decryption (Bellare & Rogaway, 2005).

Block ciphers rely on several rounds of combined operations of substitutions and permutations, called Substitution-Permutation Networks (Kam & Davida, 1979). They are used to obtain the Confusion-Diffusion Effect (Coskun & Memon, 2006), proposed by Shannon (Shannon, 1949). An example of which is the Feistel Newtork (Rayan, et al., 2016) (Ebrahim, et al., 2013).

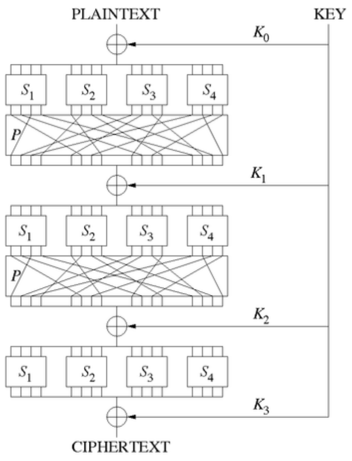


Figure 5 - A sketch of a substitution–permutation network with 3 rounds (Preneel, et al., 1998)

The goal of Confusion is to make the relationship between the key and the ciphered text as complex as possible and to make it infeasible for the cryptanalyst to find the key even if she has a large number of plaintext blocks and ciphered blocks produced with the same key. As a result, the entire ciphered text is dependent on the entire key and in different ways on different bits of the key.

The first goal of Diffusion is to dissipate the statistical structure of the plaintext over the entire ciphertext, producing a non-uniform distribution of the individual symbols (and pairs of neighboring symbols) in the plaintext. The second goal of Diffusion is to make the output bits of the ciphered text dependent in a very complex way of the bits of the input plaintext. “A single changed bit in the input plaintext will produce a change in roughly half of the bits of the ciphertext, in random positions of the ciphered text output”. This is called the Avalanche Effect (Shahzad, 2012) (Feistel, 05).

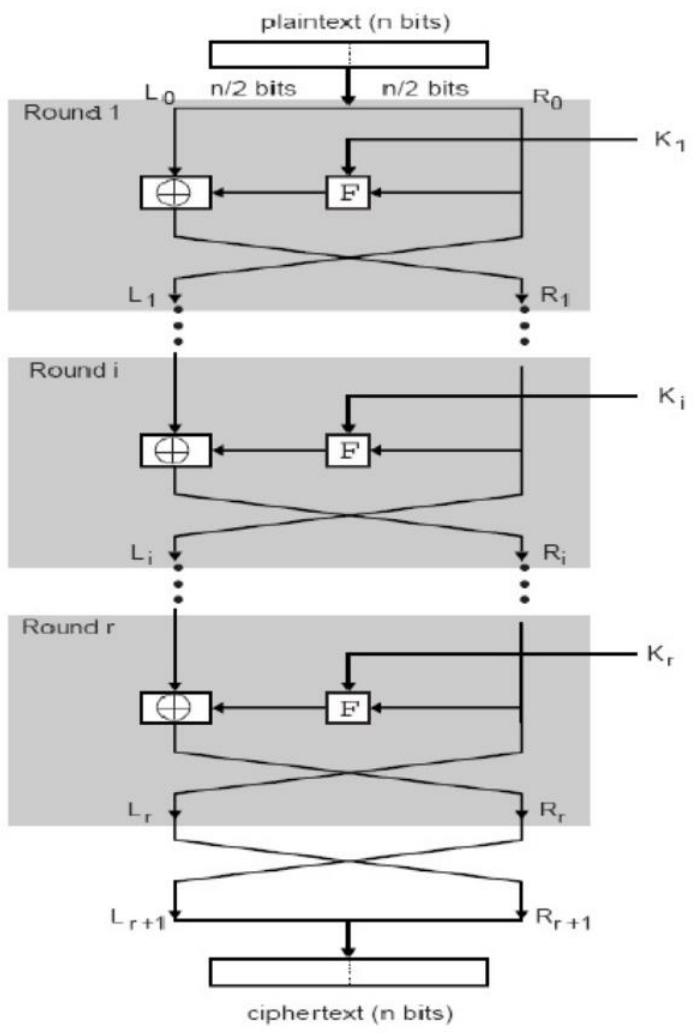


Figure 6 - Classical Feistel Network (Shahzad, 2012)

2.4.3 Block Cipher Modes of Operation

A block cipher by itself is only suitable for the secure encryption of one fixed-length block (Ferguson, et al., 2010). In order to securely encrypt messages larger than one block, a mode of operation is required. The mode of operation delineates the way the cipher's single-block operation must be applied on every block of the plaintext. As a result, block ciphers have several modes of operation. Not all APIs supports all of the modes of operation. The most commonly supported are the ECB (Electronic Code Block), the CBC (Cipher Block Chaining), the CFB (Cipher Feedback Mode), the OFB (Output Feedback) and the CTR (Counter Mode), which may be implemented in two options: Segmented Integer Counter (SIC) or Integer Counter Mode (ICM). With the exception of ECB, all other modes require an IV to function properly. Some operations modes allow direct access to any of the ciphered blocks (random access) and are parallelizable for encryption or decryption, while others are not (NIST Computer Security Division's (CSD) Security Technology Group (STG), 2012).

2.4.3.1 ECB Mode

The Electronic Code Block mode is the simplest mode of operation, but it is not safe if the message is longer than one block (Huang, et al., 2013). The message is divided in blocks of the size of the encryption algorithm and for each block of the message the encryption algorithm is applied. ECB encryption is mathematically expressed as C_i=E(K, M_i). ECB decryption is mathematically expressed as M_i = D(K, C_i). E(K, M_i) is the encryption function to encrypt the plaintext block M_i, using the key K. D(K, C_i) is the decryption function to decrypt the ciphertext block C_i, using the key K.

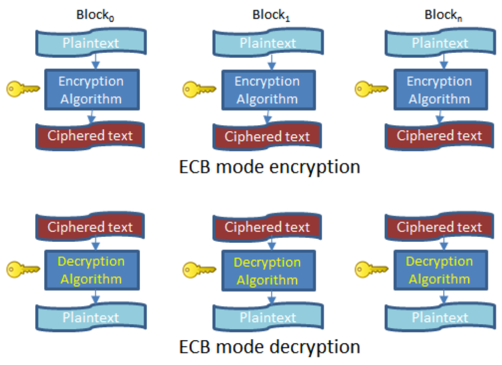


Figure 7 - ECB Mode of Operation

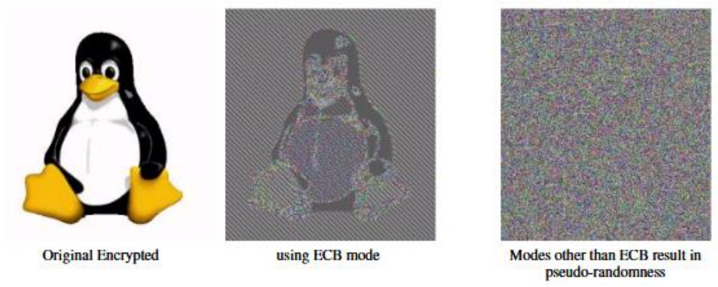


Figure 8 - Comparison between ECB and Other Modes (Huang, et al., 2013)

2.4.3.2 CBC Mode

The Cipher Block Chaining mode is more complex and safer than ECB. The cipher block depends on the result of the previous cipher block. This also prevents the encrypted message to be tampered with. If, at some block, the encrypted message is tampered with, the last block will be unrecoverable. As a result, if the last block can be successfully decrypted, the entire message has its integrity assured (Abidi, et al., 2016). CBC encryption is mathematically expressed as $C_i = E(K, M_i \oplus C_{i-1})$, where $C_0 = IV$. CBC decryption is mathematically expressed as $M_i = D(K, C_i) \oplus C_{i-1}$, where $C_0 = IV$.

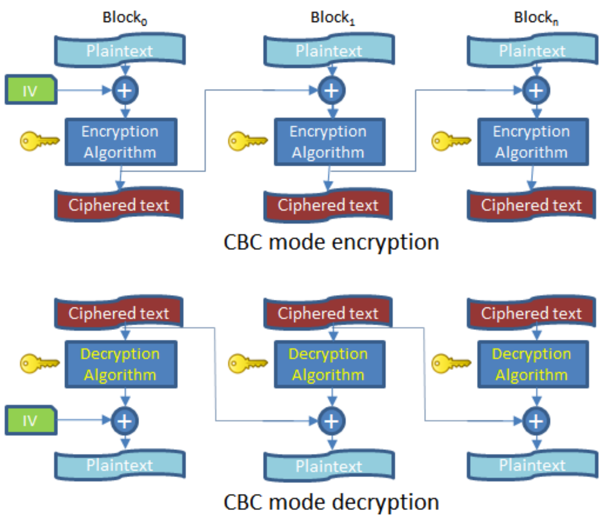


Figure 9 - CBC Mode of Operation

2.4.3.3 CFB Mode

The Cipher Feedback mode is similar to the CBC, but it transforms the block cipher algorithm into a self-synchronizing Stream Cipher. The decryption algorithm of the block cipher is not used. Only the encryption algorithm is used. Any changes in a ciphered text block will propagate to the subsequent blocks. Like in CBC mode, if the last block is recoverable, the entire message decryption is successful, thus ensuring integrity (Asmara, et al., 2017). CFB encryption is mathematically expressed as $C_i = E(K, C_{i-1}) \oplus M_i$, $C_0 = IV$, while CFB decryption is mathematically expressed as $M_i = E(K, C_{i-1}) \oplus C_i$, where $C_0 = IV$.

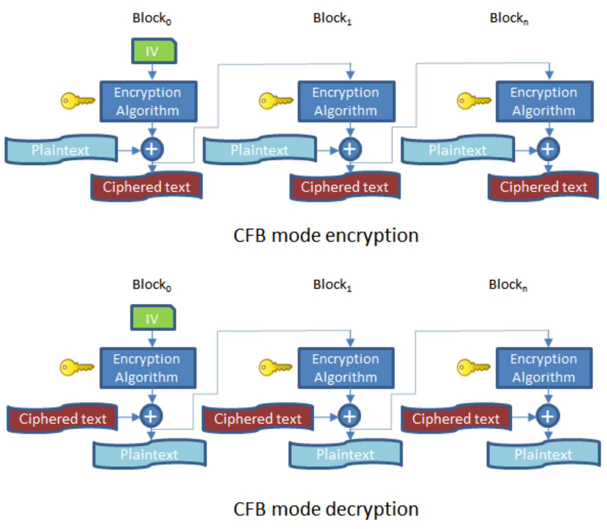


Figure 10 - CFB Mode of Operation

2.4.3.4 OFB Mode

The Output Feedback mode is another mode that transforms a block cipher algorithm into a stream cipher algorithm. The main advantage of the OFB mode is that the endpoints can generate the keystreams blocks (KS_i) in advance, since the keystreams do not depend on the plaintext. Also, if a transmission error corrupts a block or some blocks, the blocks received without errors will be decrypted and recovered (Jueneman, 1983). The disadvantages are: it is not parallelizable, it cannot be used for data integrity verification, contrary to CBC and CFB and the IV must never be repeated with the same key, even for different plaintexts. If it is repeated, all the cryptanalyst have to do is to XOR two ciphered streams to retrieve the keystream. With possession of the keystream, she will be able to decrypt any message encrypted with the same key and IV pair, without the need to know the encryption key. OFB encryption is mathematically expressed as $C_i = KS_i \oplus M_i$, $KS_i = E(K, KS_{i-1})$, $KS_0 = IV$. OFB decryption is mathematically expressed as $M_i = KS_i \oplus C_i$, $KS_i = E(K, KS_{i-1})$, $KS_0 = IV$.

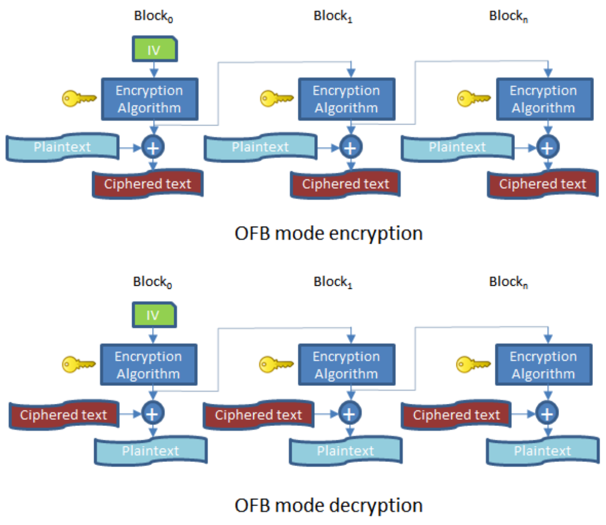


Figure 11 - OFB Mode of Operation

2.4.3.5 CTR Mode

The Counter mode, like CFB and OFB, turns a block cipher into a stream cipher. It generates the next keystream block by encrypting the next value of a counter. Even if only one bit is changed, the Avalanche Effect [2.4.2] will produce a different keystream block which is also unrelated to previous keystreams (Schneier, 1996).

When half of the block length (n/2) bits are reserved to the nonce and the other half is reserved to the counter, it is called Segmented Integer Counter (SIG). The maximum size of a message that can be encrypted, given a nonce and a key, is $2^{n/2}$ bits. For example, if the block cipher algorithm produces a block of 128 bits, the 64 leftmost bits are reserved to the nonce while the 64 rightmost bits are reserved to the counter. The first counter is 0 and the last if 2^{64}-1.

When a random number [0..2^{n-1}] is chosen as a nonce, it is called Integer Counter Mode (ICM). The maximum message size that can be encrypted is approximately 2^n. However, in ICM, when using the same key with a different nonce, one must be very careful not to allow the next counter to overlap with a counter used before with the previous nonce. For example, if the block cipher algorithm produces a block of 128 bits and the first nonce is 2^{125} -1 and the second nonce is 2^{125} + 8, the counters will overlap after 10 blocks. Overlapping counters produce the same keystream in the interval the counters overlap.

A practical example: using the Java Cipher Class encrypt a 64 Byte long string of blanks (20 Hex) twice. Set the encryption key “011234567890abcdef” (such a weak key must never be used for real life applications) for both encryptions. Let the first IV: “0000000000000000” and the second “0000000000000001”. The ciphered texts are:

30303030303030303030303030303030

7E35F53BCBC186AA3959A8204E340E4E

690EFA3BD0C256E774AD14D0740424BE

682704CAAEB97E23C2A8D8BA70301997

4C7288A1312650520B066E9D27E2AB5C

and

30303030303030303030303030303031

690EFA3BD0C256E774AD14D0740424BE

682704CAAEB97E23C2A8D8BA70301997

4C7288A1312650520B066E9D27E2AB5C

517919C6AC9E42EF9113E50E0AD4120B

respectively, converted to HEX (Note in bold where the ciphered texts overlaps).

CTR mode has been proved secure by (Bellare, et al., 1998) and (Luby & Rackoff, 1988). They argue, since CTR changes a Pseudorandom Permutation (PRP) into a Pseudorandom Function (PRF), if the PRP is proven to be secure, so must be PRF. In fact, Bellare *et al.* (Bellare, et al., 1998), argue that security is increased by making a block cipher algorithm non-invertible.

The advantages of CTR are: padding is not required; it is parallelizable for both encryption and decryption; it allows random access to a block, either for encryption or decryption, since the blocks are independent from previous ones, meaning blocks can be encrypted or decrypted independently. Like OFB, keystreams can be generated in advance. The main disadvantage of CTR is its sensitivity to usage errors (Lipmaa, et al., 2000) like a nonrandom nonce or the possibility of keystream overlapping. As a result, key management and usability with CTR is even more critical.

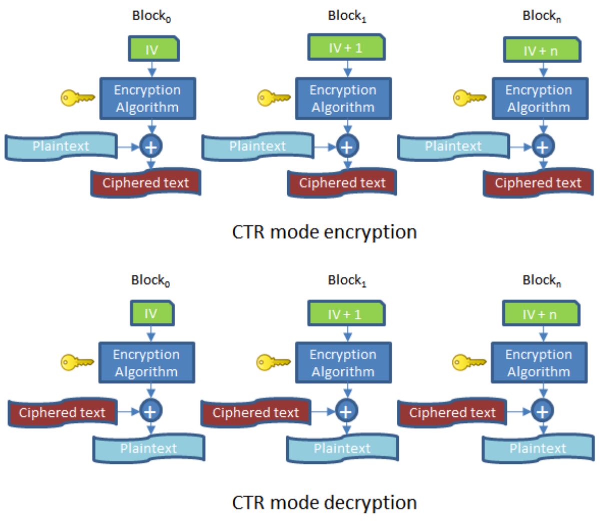


Figure 12 - CTR Mode of Operation

Each mode of operation has its advantages and disadvantages. Table 1 compares and summarizes the modes of operation. The usage sensitivity takes into account the following factors: The need for padding and the possibility of the use of an unsafe padding scheme, the need for an IV and the impact of the randomness of the IV on the security of the mode of operation. ECB and CBC require a padding scheme. ECB does not require an IV. A nonrandom IV compromises security on CBC and CFB, if the same plaintext is encrypted twice with the same IV and Key pair. However in OFB and CTR modes, if the IV is repeated, it compromises security even for different plaintexts.

Mode of Operation	Encryption Parallelizable	Decryption Parallelizable	Random Access	Requires Padding	Requires IV	Usage Sensitivity
ECB	Yes	Yes	Yes	Yes	No	Low
CBC	No	Yes	Yes	Yes	Yes	High
CFB	No	Yes	Yes	No	Yes	Medium
OFB	No	No	No	No	Yes	Very High
CTR	Yes	Yes	Yes	No	Yes	Very High

Table 1 - Comparison of the Most Common Block Cipher Modes of Operation

2.4.4 Comparison of Symmetric Key Encryption Algorithms

2.4.4.1 Data Encryption Standard (DES)

DES was designed by IBM in 1973-1974 based on their Lucifer cipher. It was the first encryption standard to be published by the NIST in 1975. The DES uses the Feistel Network, operates with an effective key length of 56 bits, producing an output block of 64 bits (Ebrahim, et al., 2013). Although the key is 64 bits long, 8 bits are parity (bits 8, 16, 24, 32, 40, 48, 56 and 64). The encryption function has 16 iterations or rounds. From the key, the algorithm schedules a 48 bit subkey to be used for each iteration (Dahab & López, 2007). The best attack against DES is $2^{39} – 2^{43}$ time with 2^{43} known plaintexts (data) (Pascal, 2001). However a brute force attack costs 2^{56}, which is feasible to execute on modern computers. DES requires up to 6 parameters: the plaintext, the encryption key, the operation mode, the padding scheme, the IV and the function to be used (encryption or decryption). When using OFB or CTR modes, a padding scheme is not necessary.

2.4.4.2 Triple-DES

DES was superseded by triple DES (3DES) in November 1998, concentrating on the noticeable imperfections in DES without changing the original structure of DES algorithm. 3DES applies DES three times. The first step uses DES encryption with a 56 bit key (K_1), the second step uses DES decryption with a second 56 bit key (K_2) and the third step uses the DES encryption with a third key (K_3). As a result, 3DES uses a 168bit key, even though it is possible to define a 112 bit key by making $K_3 = K_1$ (Ebrahim, et al., 2013). If $K_1=K_2=K_3$, 3DES becomes DES.

The best attack against 3DES is 2^{113} time, with 2^{32} data and 2^{88} memory (data needed for computations). Even if it seems infeasible at a first look, 3DES is also vulnerable to the SWEET32 attack, making it possible to decrypt OPEN-Vpn or TLS traffic after collecting $2^{36.6}$ blocks, approximately 785GB. The authors of SWEET32 were able to break the cipher between 18.6 and 30.5 hours (Bhargavan & Leurent, 2016). As a result of the SWEET32 attack, VPN and TLS implementations are removing 3DES from their cipher suite. 3DES requires up to 6 parameters: the plaintext, the encryption key, the operation mode, the padding scheme, the IV and the function to be used (encryption or decryption). When using OFB or CTR modes, a padding scheme is not necessary.

2.4.4.3 IDEA

The International Data Encryption Algorithm (IDEA) was developed in 1990. It uses a 128 bit key and a 64 bit block. As opposed to DES or 3DES, IDEA does not use the Feistel Network. It uses a substitution-permutation transformation instead. Each round uses 6 16-bit subkeys, while the half-round uses 4, a total of 52 for 8.5 rounds. The first 8 sub-keys are extracted directly from the key, with the first subkey from the first round being the lower 16 bits. The subsequent groups of 8 subkeys are created by rotating the main key left 25 bits between each group of 8. As a result, the key is rotated less than once per round, on average, for a total of 6 rotations (Lai & Massey, 1990).

The best attack against IDEA is $2^{126.1}$ time, still infeasible by today's computational power (Khovratovich, et al., 2012). However the key size and the short block length justifies IDEA not to be a good choice for future applications. Please note that the SWEET32 attack can be applied with any 64bit block length encryption algorithm. IDEA requires up to 6 parameters: the plaintext, the encryption key, the encryption mode, the padding scheme, the IV and the function to be used (encryption or decryption). When using OFB or CTR modes, a padding scheme is not necessary and only the encryption function is used.

2.4.4.4 Blowfish

Blowfish is also a symmetric key Feistel Structured algorithm consisting of 2 parts: key expansion part and data-encryption part. Blowfish is a block cipher that uses a 64 bit block with 16 rounds, allowing a variable key length from 32-448 bits (Ebrahim, et al., 2013). The key expansion function is a hash algorithm, which produces 18 arrays of 32-bit subkeys and 4 S-Box arrays of 32-bit with 256 entries each, totaling 4168 Bytes. The algorithm was designed to accept two modes of operation: ECB or CBC. Blowfish was designed by Bruce Schneier in 1993 and offered free for public use (Schneier, 1993).

Although no attack was successful against the 16 rounds of the Blowfish algorithm (Rijman, 1997), it is also vulnerable to the SWEET32 attack (Bhargavan & Leurent, 2016). Blowfish requires up to 6 parameters: the plaintext, the encryption key, the encryption mode, the padding scheme, the IV and the function to be used (encryption or decryption). The key length is flexible. In ECB mode, an IV is not necessary.

2.4.4.5 Twofish

Twofish is 16 round Feistel Network algorithm designed by Bruce Schneier *et al.*, in 1998. Towfish was one of the five finalists of the Advanced Encryption Stantard contest (1997-2000). Twofish uses a 128 bit block cipher and accepts keys of 128, 192 and 256-bit lengths (Schneier, et al., 2000).

So far, no attack was successful against the 16 rounds of the Twofish algorithm or its key schedule algorithm (Ferguson, 1999). Twofish requires up to 7 parameters: the plaintext, the encryption key, the encryption key length, the encryption mode, the padding scheme , the IV and the function to be used (encryption or decryption). However, Unlike Blowfish, the key length is not flexible. When using OFB or CTR modes, a padding scheme is not necessary and only the encryption function is used.

2.4.4.6 Serpent

Serpent is a 32 round Substitution-Permutation Network algorithm with a 128 bit block length and 128, 192 or 256-bit key lengths. Designed by Ross Anderson *et al.*, in 1998, Serpent made second place of the Advanced Encryption Standard contest (1997-2000), even though it is safer than Rijndael, the winner (Anderson, 1999).

Rijndael was chosen because it was significantly faster than Serpent (up to 14 rounds, depending on the key size, against the 32 rounds of Serpent) (Schneier, et al., 2000).

So far, no attack was succesfull against the 32 rounds of the Serpent algorithm or its key schedule algorithm (Biham, et al., 2001). Serpent requires up to 7 parameters: the plaintext, the encryption key, the encryption key length, the encryption mode, the padding scheme, the IV and the function to be used (encryption or decryption). When using OFB or CTR modes, a padding scheme is not necessary and only the encryption function is used.

2.4.4.7 Rijndael (AES)

Rijndael is a Substitution-Permutation Network algorithm with a 128 bit block length and 128, 192 or 256-bit key lenghts. Depending on the key length it uses 10, 12 or 14 rounds respactively. Designed by Vincent Rijmen and Joan Daemen, in 1998, Rijndael was the winner of the Advanced Encryption Stantard contest (1997-2000) (Huang, et al., 2013). Rijndael derives subkeys from the key, using a key scheduling algorithm. It requeires a 128-bit key for each round of the block encryption. A High-level description of the Rijndael algorithm consists of (Dahab & López, 2007):

1. Key expansion
2. Initial round: "AddRoundKey" – each byte of the state is combined with a block of the round key using bitwise XOR.
3. Next rounds
    a. "SubBytes" a non-linear substitution step where each byte is replaced with another according to a lookup table.
    b. "ShiftRows" a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
    c. "MixColumns" a mixing operation which operates on the columns of the state, combining the four bytes in each column.
    d. "AddRoundKey"

4. Final round
    a. "SubBytes" a non-linear substitution step where each byte is replaced with another according to a lookup table.
    b. "ShiftRows" a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
    c. "AddRoundKey"

Rijndael is recommended by NIST, NSA, CRYPTREC and NESSIE. It is fast and its has been implemented in the hardware of most modern processors (Intel Corporation, s.d.) (IBM Crypto Development Team, 2015) (Grisenthwaite, s.d.).

In 2009, bugs were found in Rijndael key scheduling algorithm, resulting in the decrease of its complexity. In 2011, the attack was improved. The best attacks for AES are:

AES-128	$2^{126.1}$ time	2^{88} known plaintexts	2^{8} memory
AES-192	$2^{189.7}$ time	2^{80} known plaintexts	2^{8} memory
AES-256	$2^{254.4}$ time	2^{40} known plaintexts	2^{8} memory

Table 2 - Best Attack Complexities for AES

AES requires up to 7 parameters: the plaintext, the encryption key, the encryption key length, the encryption mode, the padding scheme, the IV and the function to be used (encryption or decryption). When using OFB or CTR modes, a padding scheme is not necessary and only the encryption function is used.

2.4.4.8 RC4

RC4 is a stream cipher designed in 1987 by Ron Rivest for RSA Security. It accepts a variable key size, from 8 to 2048 bits, with byte-oriented operations.The algorithm is very simple, fast and easy to use. The initial state is calculated from the encryption key, meaning the same message encrypted with the same key will produce the same ciphered text. However it requires only two parameters, the plaintext and the key for encryption or the ciphered text and the key for decryption. RC4 works in trhee stages (Stallings, 2011):

1) A state array of 256 Bytes is initialized using the key using the folowwing algorithm:

```
j = 0;
for i = 0 to 255:
   S[i] = i;
for i = 0 to 255:
   j = (j + S[i] + K[i]) mod 256;
   Swap S[i] and S[j];
```

The key is no longer used, after the state array is initialized. The second step is the initial permutation of the state array S.

```
j = 0;
for i = 0 to 255 do
   j = (j + S[i] + T[i]) mod 256;
   Swap (S[i], S[j]);
```

The final step is the keystream generation, one Byte at a time and a Bitwise XOR operation with the plaintext. After the keystream is generated, the state is modified.

```
i, j, k = 0;
while (k<messageLenght)
   i = (i + 1) mod 256;
   j = (j + S[i]) mod 256;
   Swap (S[i], S[j]);
   t = (S[i] + S[j]) mod 256;
   KS = S[t];
   c[k] = M[i] ⊕ KS;
return c;
```

The state is expected to repeat itseft every 2^{1024} times for the same key. Such an enourmous keystream space made RC4 one of the most popular and secure algorithms of the 1990s and the initial half of the 2000s. However, in 2007, Vanhoef and Piessens discovered biases in the keystream generation, making it possible to derive the encryption key after $9{\cdot}2^{27}$ ciphertexts captured with a 94% success probability. TLS, Microsoft RDP, WEP and WPA-TKIP encryption protocols, which used RC4, have all been compromised (Vanhoef & Piessens, 2015). The first two protocols removed RC4 from their cipher suites. However, the last two wireless encryption protocols, which are RC4 based, became obsolete with the best attack costing 2^{20} time and $2^{16.4}$ data with a 95% probability (Tews, et al., 2007). RC4 requires only 2 parameters: the plaintext and the encryption key, which is flexible.

2.4.4.9 HC-256

HC-256 is a stream cipher algorithm and is one of the four finalists of the eSTREAM contest (software profile). HC-256 uses a 256 bit key and an IV (nonce) of 256 bits with a word size of 32 bits. The initialization process consists expanding the key and the IV into 2 arrays P and Q, each containing 1024 elements of 32 bit integers. Like RC4, after each keystream generated, the state of P and Q are updated. After 4096 steps of key generation without outputting the keystreams, the cipher is ready to produce keystreams to be XORed with the plaintext. The 65547-bit state of HC-256 ensures de period of the keystream to be huge. The authors estimate the period to be about 2^{66546} (Wu, 2004).

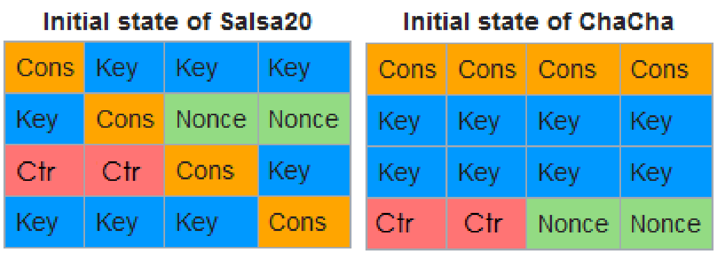
The best attack against HC-256 takes about $2^{276.8}$ linear equations involving binary keystream variables (Sekar G, 2009). HC-256 requires 3 parameters: the plaintext, the encryption key and the IV. The key is not flexible.

2.4.4.10 Salsa20 and ChaCha

Salsa is another of the four finalists of the eSTREAM contest (software profile) stream cipher algorithm. It uses a 256 bit key, a 512 bit state and 20 rounds based on an ARX (add-rotate-xor) structure (Mahfouf, et al., 2002). It also requires a 64 bit counter and a 64bit nonce. Salsa20 expands a 256-bit key and a 64-bit nonce (unique message number) into a 270 Byte stream. It outputs 64 Bytes (512 bits) as the keystream and discards the last 6 Bytes of the generated Byte stream. Like HC-256, Salsa20 uses arrays of 32bit words (Bernstein, 2008).

At this time, there is no known attack against the 20 rounds of the Salsa20 algorithm. In fact, in 2013, Mouha and Preneel published a proof that 15 rounds of Salsa20 was 128-bit secure against differential cryptanalysis. They calculated there is no differential characteristic with higher probability than 2^{-130}. As a result, differential cryptanalysis would be more costly than 128-bit key exhaustion (Mouha N, 2013). With reduced rounds (8/20) the best attack to recover the secret key is 2^{255} operations, using $2^{11.37}$ keystream pairs (Tsunoo Y, 2007). Salsa20 requires 3 parameters: the plaintext, the encryption key and the nonce. The key is not flexible.

ChaCha is a variant of the Salsa algorithm. The difference is in the initial state (Bernstein, 2008).



Initial state of Salsa20

Cons	Key	Key	Key
Key	Cons	Nonce	Nonce
Ctr	Ctr	Cons	Key
Key	Key	Key	Cons

Initial state of ChaCha

Cons	Cons	Cons	Cons
Key	Key	Key	Key
Key	Key	Key	Key
Ctr	Ctr	Nonce	Nonce

Figure 13 - Initial States of Salsa20 and ChaCha

2.4.4.11 Comparison Summary

In this section, we listed the most used and the most advanced symmetric encryption algorithms. The table below summarizes the characteristics of the algorithms, if it is safe to use at this time and in the near future. The term “safe”, as it was defined in the introduction section, means the algorithm cipher is secure and therefore, reliable. The algorithm is neither broken nor significantly weakened. The last column refers to our classification based on the number and complexity of the parameters and flexibility of key lengths in compliance with the 6th principle of Kerckhoff [2.1]. Parameters are not necessarily arguments of a function or method, but rules required to be known by the user. Each required parameter is given a point. If the algorithm is a stream cipher or can be converted to a stream cipher algorithm via CTR or OFB mode of operation and the algorithm specification leave it up to the developer or user to set the IV, another point is added to the algorithm as a penalty due the fact a constant key and IV will produce the same keystream every time, severally weakening security. If the algorithm accepts variable key lengths, a point is taken from the algorithm as a reward. 10 minus the total points is the final value of the usability index. The highest the Usability Index, the more an encryption algorithm adheres to the 6th principle of Kerckhoff. Usability Indexes in the following are worst cases.

Algorithm	Block Cipher or Stream Cipher	Block Size	Maximum Key Length	Safe or Unsafe to Use	Usability Index
DES	Block	64	56	Unsafe	4
3DES	Block	64	168	Unsafe	4
IDEA	Block	64	128	Unsafe	4
Blowfish	Block	64	448	Unsafe	5
Twofish	Block	128	256	Safe	3
Serpent	Block	128	256	Safe	3
AES	Block	128	256	Safe	3
RC4	Stream	NA	2048	Unsafe	8
HC-256	Stream	NA	256	Safe	7
Salsa20	Stream	512	256	Safe	7

Table 3 - Symmetric Key Algorithms Comparison

RC4 is the algorithm that more adheres to the 6th principle of Kerckhoff, requiring only the plaintext and the key, which has a flexible length.

2.5 Collision Resistant Hash Functions (CRHF)

Collision Resistant Hash Functions (CRHF) are used for various applications such as message authentication, digital signatures, pseudorandom bit generation, integrity assurance and others (Akhimullah & Hirose, 2016). Hashes are mathematical functions that compress an input of arbitrary length to a result with a fixed length. Hash functions are also used to allocate, as uniformly as possible, storage for the records of a file. CRHF are hashes which collisions are hard to find.

The formal definition of a collision resistant hash function is credited to Damgård (Damgard, 1988).

A collision resistant hash function is a function h satisfying the following conditions (Preneel, 2003):

1. The description of h must be publicly known and should not require any secret information for its operation (*extension of Kerckhoffs's principles*) [2.1].

2. The argument X can be of arbitrary length and the result h(X) has a fixed length of ***n*** bits. In mathematical language: $h: \{0, 1\}^* \rightarrow \{0, 1\}^n$

3. Given h and X, the computation of h(X) must be "easy".

4. The hash function must be one-way in the sense that given a Y in the image of h, it is “hard” to find a message X such that h(X) = Y and given X and h(X) it is “hard” to find a message X’ ≠ X such that h(X’) = h(X).

5. The hash function must be collision resistant: this means that it is “hard” to find any two distinct messages that hash to the same result.

In order to satisfy conditions 4 and 5, a CRHF must be resistant to preimage, second preimage and collision attacks (Rogaway & Shrimpton, 2004). For a preimage or a second preimage attack to be successful (4), the computational complexity required is $\boldsymbol{O(2^n)}$, while for collisions (5), the computational complexity required is $\boldsymbol{O(2^{n/2})}$.

Preimage and second preimage resistance also implies:

6. Pseudo-Randomness: Output of h meets standard tests for pseudorandomness.

7. Non Malleability: given h(X), it is infeasible to produce h(X’) where X and X’ are “related” in any fashion. Eg.: X’ = X+1, X’ = f(X).

2.5.1 Construction Structures of Collision Resistant Hash Functions

In order to produce a fixed length output for a variable length input, the input string is divided into a series of blocks and the compression function is called iteratively for each of the blocks. The internal state is updated each iteration and the final state is either the hash output or the parameters for the finalization function which produces the hash output. The way the internal state is updated is called the construction structure. Since blocks have a unique length, the final block is always padded, even if it has the same length of the block. The padding scheme is usually a 1 bit followed by as many 0 bits as it is needed to complete the block length minus the bits reserved for the message size. Putting the message size into the final block helps to avoid collisions of two different size strings, which would produce the same hash output (STALLINGS, 2011). An adversary must either find two messages of equal length that hash to the same value or two messages of differing lengths which, together with their length values, hash to the same value.

The most used construction structure for hash algorithms is the Merkle-Damgård. It is so popular, because Merkle and Damgård independently proved the construction structure was secure if an appropriate padding scheme is used and the compression function is collision-resistant (Damgård, 1989) (Merkle, 1989). The construction uses a fixed IV (initial value) of n bits as the initial state and partitions the input string into L fixed size blocks of *b* bits, where $b \geq n$. The compression function *f* combines the state of n bits with the *b* bits from the input string and updates the state. L iterations are necessary to produce the final state. The finalization function consists of compressing the padding with the last state, which will produce the hash output.

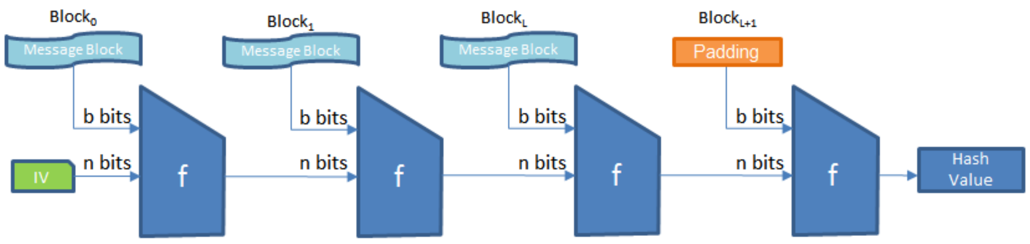


Figure 14 - The Merkle-Damgård Construction

The Merkle-Damgård construction has several vulnerabilities (Coron, et al., 2005), particularly the Length Extension Attack [2.7.2] and the Multicollisions Attack (Joux, 2004).

There are other construction structures, some made of block cipher encryption algorithms (Bartkewitz, 2009). Of those, one that is relevant to this work is the Miyaguchi–Preneel construction, used as the mode of operation of the encryption algorithm of the Whirlpool CRHF.

Each block of the message is encrypted by the encryption algorithm. The output ciphertext is then bitwise XORed with the same message block and then also bitwise XORed with the previous hash value to produce the next hash value. The previous hash value is passed as a parameter to a function g to be converted or padded to fit as the key for the encryption algorithm. The first value to be passed to g is the constant IV (Shrimpton & Stam, 2008).

In mathematical notation, the Miyaguchi–Preneel construction is described as:

$$H_i = E(g(H_{i-1}),m_i) \oplus H_{i-1} \oplus m_i,\ H_0 = IV,\ H_L = Hash$$

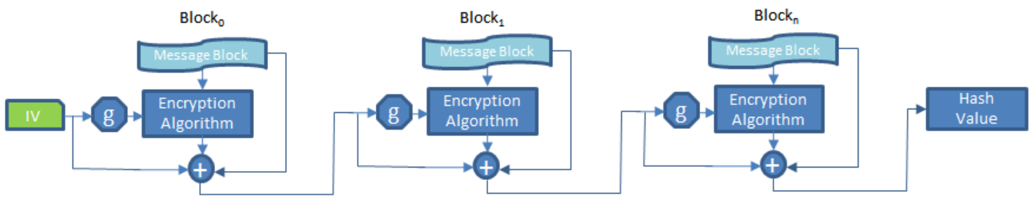


Figure 15 - Miyaguchi–Preneel Construction

Another relevant construction structure for our work is the Sponge Construction (Bertoni, et al., 2007), used by the SHA-3 CHRF. The Sponge Construction is very complex, slow in performance when compared to other constructions (Dahal, et al., 2013) but is proven to be superior in terms of security. The Sponge Construction is resistant to Length Extensions Attacks (Bertoni, et al., 2009).

The Sponge Construction contains the following components:

1) A state memory, S, containing b bits,
2) A function $f : S \rightarrow S$ which transforms the state memory (often it is a pseudorandom permutation of the 2^b state values)
3) A padding function *P*

The Sponge Construction operates according to the following steps:

a) The state S is initialized to zero
b) The input string is padded. The input is transformed into blocks of *r* bits using the padding function *P*.
c) For each r-bit block B of the padded input:

R is replaced with R ⊕ B
S is replaced by *f*(S)

Those three steps make up "absorbing" phase of the sponge. The sponge function output is now ready to be produced ("squeezing" phase) as follows:

The first *r* bits of the state are returned as output blocks, interleaved with calls to the function *f*. The number of iterations is determined by the requested number of bits. Finally the output is truncated to the requested length (Bertoni, et al., 2011).

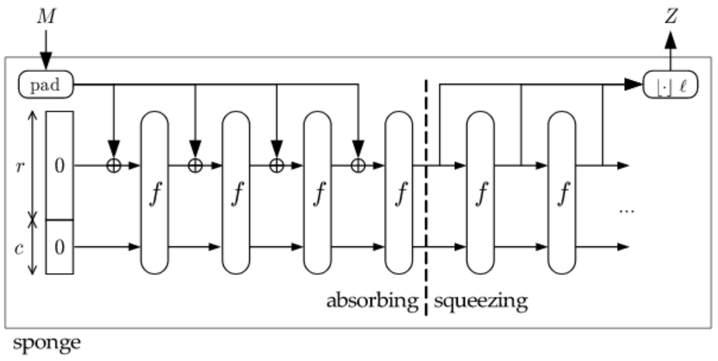


Figure 16 - The Sponge Construction (Bertoni, et al., 2011)

2.5.2 MD5

Message Digest 5 (MD5) was designed by Ronald Rivest from RSA in 1991. MD5 uses the Merkle-Damgård construction with 512 bit block length and an output of 128 bits. The compression function has 4 rounds. The security of MD5 is compromised and it is no longer suitable for digital signatures, integrity validation and password storage (Dougherty, 2008). However it is still useful to authenticate messages. RFC-6151 specifies the recommended uses for MD5 (Turner & Chen, 2011).

The best attack against MD5 cost only 2^{18} time and takes only seconds in a regular PC (Xie, et al., 2013). Preimage resistance is $2^{123.4}$ for a full preimage (Sasaki & Aoki, 2009). Even if MD5 has not been broken, The Birthday Attack would take only 2^{64} hashes to find a collision pair.

2.5.3 SHA-1

Secure Hash Algorithm 1 was designed by the NSA in 1993. It produces a 160 bit output with an input block of 512 bits. It uses the Merkle-Damgård construction and its compression function takes 80 rounds (NIST - National

Institute of Standards and Technology, 2015). SHA-1 was the main algorithm used for digital signatures until 2016 and was hardware implemented in many CPUs (Intel Corporation, 2013) (Mitchell & Kim, 2017) (ARM, 2015).

In 2011, Marc Stevens published a paper, where collisions could be found with a complexity between $2^{60.3}$ and $2^{65.3}$. However, the first public collision was published by CWI (Centrum Wiskunde & Informatica) and Google Research in February, 2017. According to the authors, the attack required "the equivalent processing power as 6,500 years of single-CPU computations and 110 years of single-GPU computations" (Stevens, et al., 2017). As a result, Microsoft, Google, Apple and Mozilla have all announced that their respective browsers stopped accepting digital certificates signed with SHA-1 by the end of 2017 (Cloutier & Vignesh, 2015) (Apple Support, 2017) (Google, 2014) (Mozilla Security Blog, 2014).

2.5.4 RIPEMD-160

RACE Integrity Primitives Evaluation Message Digest (160 bits) was developed in 1996 by Hans Dobbertin, Antoon Bosselaers and Bart Preneel. It uses the Merkle-Damgård construction and its compression function takes 80 rounds. The input block length is 512 bits and the output digest length is 160 bits (Dobbertin, et al., 1996). RIPEMD-160 is certified by CRYPTREC (The Ministry of Internal Affairs and Communication of Japan and The Ministry of Economy, Trade and Industry of Japan, 2003).

No known attack has been published against the full 80 rounds of RIPEMD-160. The best attack is able to find collisions on 31 out of 80 rounds (Ohtahara, et al., 2010).

2.5.5 SHA-256, SHA-384 and SHA-512

Secure Hash Algorithm 2 is a family of CRHF designed by the NSA. It was first published by NIST as a standard in 2001. The family also includes a 224 bit algorithm. The compression function is virtually the same for the entire family except for the addition operation, which is A + B mod 2^{32} for SHA-256 and A + B mod 2^{64} for SHA-512 or SHA-384. The IVs, the number of rounds, the input block length and, obviously, the output digest length differ. The SHA-2 family of CRHF uses the Merkle-Damgård construction with 64 rounds for SHA-256 and 80 rounds for SHA-384 or SHA-512. The length of the input block is 512 bits for SHA-256 and 1024 for SHA-384 or SHA-512. The maximum message size is 2^{64} -1 bits for SHA-256 and 2^{128} -1 bits for SHA-384 or SHA-512. SHA-384 is obtained by truncating the left-most 384 bits of a 512 bit HASH output (NIST - National Institute of Standards and Technology, 2015).

Like, SHA-1, SHA-256 have been hardware implemented in most recent CPUs (Intel Corporation, 2013) (Mitchell & Kim, 2017) (ARM, 2015).

At this time, there are no attacks capable of reducing preimage, 2nd preimage or collision resistance. The best attacks against SHA-256 and SHA-512 are 43 of 64 rounds taking $2^{254.9}$ time (2^{6} memory) and 46 of 80 rounds taking $2^{511.5}$ time (2^{6} memory) (Aoki, et al., 2009) respectively for preimage attacks. For collision attacks, the best attack is 43 of 64 rounds taking $2^{254.9}$ time (2^{6} memory) and 46 of 80 rounds taking $2^{511.5}$ time (2^{6} memory), respectively (Aoki, et al., 2009).

2.5.6 Whirlpool

Whirlpool is a 512 bit input block and digest size, designed by Paulo Barreto and Vincent Rijmen. It uses a Merkle-Damgård construction and a 10 round modified version of the AES cipher, using the Miyaguchi–Preneel construction as the mode of operation. Although it is slower then SHA-512, Whirlpool accepts messages lengths up to 2^{256} -1 bit. The IV is a 512 bit long string of Zeros. Whirlpool uses a padding and the message length as the parameter for the finalization function. Whirlpool is certified by NESSIE, ISO and IEC (Barreto & Rijmen, 2000).

At this time, there are no attacks capable of weakening the full 10 rounds of Whirlpool, either for reducing the preimage resistance or for reducing the collision resistance. The best attack against Whirlpool to reduce collision resistance is 4.5 rounds taking 2^{120} time (Mendel, et al., 2009).

2.5.7 SHA-3

Keccak (Secure Hash Algorithm 3) is the latest member of de SHA Family of standards, released by NIST in 2015. Keccak was the winner of the SHA-3 contest and is based in the Sponge Construction [2.5.1]. SHA-3 possible output digests lengths are: 224, 256, 384 and 512 bits. The respective input block lengths are: 1152, 1088, 832 and 576 bits (1600 – (2 x Digest Size)). The internal state of Keccak is always 1600 bits. The number of rounds is 24, no matter the digest size (Bertoni, et al., 2013).

SHA-3 is immune to the Message Length Attack, meaning it would be more costly than 2^n to succeed in such an attempt (Bertoni, et al., 2011). However, it is considerably slower than SHA-2 in software implementation (Dahal, et al., 2013).

At this time, no attacks have been successful against SHA-3. The best attack against SHA-3-256 is limited to 5 rounds at the cost of 2^{115}. Against SHA-3-384, the attack is limited to 4 rounds, costing 2^{147}. For SHA-3-512, the attack is limited to 3 rounds, with the cost of approximately 2^{34} (Dahal, et al., 2013) (Dinur, et al., 2013).

2.5.8 Skein

Skein was one of the 6 finalists of the SHA-3 contest. It is the fastest of the finalists, being faster and safer than SHA-2. Skein was designed to replace any and all of the previous mentioned hashes, except Keccak. Skein can produce digest sizes of 128 (MD5), 160 (SHA-1, RIPEMD-160), 224 (SHA-224), 256 (SHA-256), 384 (SHA-384), 512 (SHA-512) and 1024 bits. Skein uses UBI (Unique Block Iteration), a construction variant of the Matyas-Meyer-Oseas structure. UBI requires a configuration block which parameterizes the hash and the encryption algorithm (tweak). The finalization function is also a call or several calls to the UBI construction, allowing skein to generate the desired output size (Ferguson, et al., 2010). The encryption algorithm is the Threefish tweakable block cipher which can produce ciphertexts of 256, 512 or 1024 bits, using an ARX structure (Mahfouf, et al., 2002).

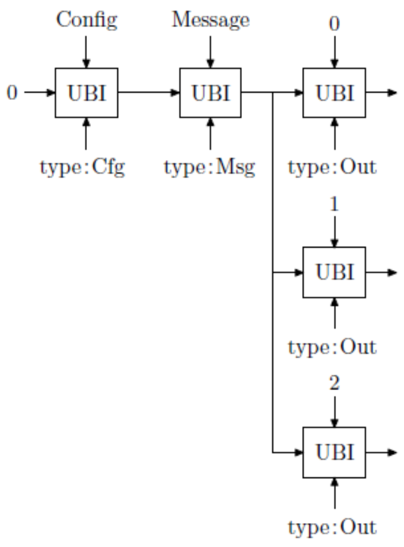


Figure 17 - Skein UBI Producing Larger Output Sizes (Ferguson, et al., 2010)

Even though Skien is immune to the Length Extension Attack [2.7.2], the authors created a personalized form of authentication: Skein-MAC. It first consumes the key, passed as a configuration parameter and then the message.

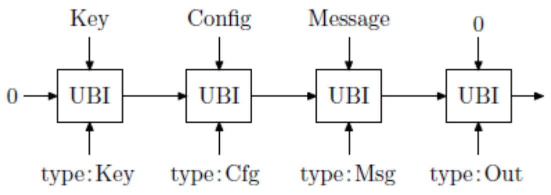


Figure 18 - Skein – MAC

The best attack against Skein 256 can find collisions up to 53 of 72 rounds. For Skein 512, collisions can be found up to 57 of 72 rounds. No attack has been published, at this time, for Skein 1024, which takes 80 rounds. The cost for 42 rounds of Skien-256 is 2^{244} and for 46 rounds of Skien-512 is 2^{495} (Khovratovich, et al., 2010).

2.5.9 Collision Resistant Hash Functions Summary

In this section, we listed the most used and the most advanced CRHFs. The table below summarizes the characteristics of the algorithms, number of rounds broken (rounds where collisions can be produced) and if it is safe to use at this time and in the near future for message integrity validation and digital signatures (according to the NIST) [Figure 1].

Algorithm	**Block Size**	**Digest Size**	**Length Extension Attack**	**Broken Rounds**	**Percent Broken**	**Safe or Unsafe for Use Now**	**Safe or Unsafe for Future Use**
MD5	512	128	Applies	4/4	100	Unsafe	Unsafe
SHA-1	512	160	Applies	80/80	100	Unsafe	Unsafe
RIPEMD-160	512	160	Applies	48/80	60	Safe	Unsafe
SHA-256	512	256	Applies	31/64	48	Safe	Unsafe
SHA-384	1024	384	Applies	24/80	30	Safe	Unsafe
SHA-512	1024	512	Applies	24/80	30	Safe	Safe
SHA-3-256	1088	256	Does not Apply	5/24	21	Safe	Unsafe
SHA-3-512	576	512	Does not Apply	3/24	12,5	Safe	Safe
Whirlpool	512	512	Applies	4.5/10	45	Safe	Safe
Skein-256	256/512	256	Does not Apply	53/72	74	Safe	Unsafe
Skien-512	512/1024	512	Does not Apply	57/72	79	Safe	Safe
Skein-1024	1024	1024	Does not Apply	NA	NA	Safe	Safe

Table 4 - CRHF Comparison for Message Integrity and Digital Signature

2.6 Message Authentication

A CRHF ensures integrity but not authenticity. There are several ways a key a message and a CRHF can be combined as a function in order to guarantee integrity and authenticity.

A message authentication scheme must satisfy the following conditions:

1. The description of the MAC must be publicly known and the only secret information lies in the key K (extension of Kerckhoffs' principles).

2. The message to be authenticated M can be of arbitrary length and the result MAC(K, M) has a fixed length of n bits.

3. Given a MAC, M and K, the computation of MAC(K, M) must be "easy".

4. Given a MAC and M, but not K, it must be "hard" to determine MAC(K, M) with a probability of success "significantly higher" than $1/(2^{n/2})$. Even when a large set of pairs $\{M_i, MAC(K, M_i)\}$ is known, where the M_i have been selected by the opponent, it must be "hard" to determine the key K or to compute MAC(K,

M’) for any M’ $\neq M_i$. This last attack is called an adaptive chosen plaintext attack (Preneel, 2003).

2.6.1 Message Authentication Code (MAC)

The first mechanism designed to ensure both integrity and authenticity is the MAC. MAC is simply a secret key concatenated with the message to be authenticated. The concatenated string in then hashed and the digest is sent in the open with the message. The receiver, which also holds the secret key, concatenates the message received with the key and applies the same hash algorithm. If the calculated hash matches the received hash, the message is authentic, meaning it was neither tampered with nor corrupted during transmission (Tsudik, 1992). The key may be the message’s prefix or suffix.

As mentioned in [2.5.1], CRHFs which use the Merkle-Damgård construction cannot be used with this authentication method.

2.6.2 Hash Based Message Authentication Code (HMAC)

HMAC is a form of MAC that resists the Length Extension Attack. HMAC is formally defined in RFC 2104 (Krawczyk, et al., 1997): HMAC = $H((K \oplus opad) \parallel H((K \oplus ipad) \parallel M))$. The opad is the outer padding with the value 5C HEX repeated |B| times, where |B| is the size of the output hash in Bytes. The ipad is the inner padding with the value 36 HEX, also repeated |B| times. The disadvantage of HMAC is that the compression function is used at least three times. If the key size is longer than |B|, HMAC must first hash the key, producing K’. K’ then will replace K in the HMAC formula. In this case, HMAC uses, at least, four compression function calls.

HMAC can be used with any CRHF and it has been proved secure (Kim, et al., 2006) (Bellare, 2006), even for MD5 (Turner & Chen, 2011). The best key recovery attack against MD5-HMAC is 2^{97} time, 2^{89} table and 2^{97} memory, with 87% success probability (Wang, et al., 2009), infeasible to execute, satisfying MAC prerequisites. For Whirlpool, the best attack to produce collisions is 2^{256} time, 2^{256} memory and 2^{256} data (Guo, et al., 2013), meaning there is no optimization other than the birthday attack. The best key recovery attack against 6 out of 10 rounds of the HMAC-Whirlpool is 2^{496} time, 2^{448} memory and 2^{384} data (Guo, et al., 2013).

The compression function is used at least three times with HMAC. There will be at least two iterations in the inner hash and one in the outer hash. Also, there is the cost two bitwise XORs, besides the paddings.

2.6.3 ENVELOPE

ENVELOPE can be twice as fast as HMAC, depending on the message and key size, and algorithm's input block size. ENVELOPE is accepted as a secure MAC even for MD5 (Metzge & Simpson, 1995) and SHA-1 (Metzge & Simpson, 1995). The idea of the "envelope" is to pad the key in the same manner the message is padded for the hash algorithm, making it infeasible for the attacker to execute the Length Extension Attack.

ENVELOPE is defined as $H(K \parallel \pi_K \parallel M \parallel K)$. Please note that a second padding will be made automatically by any hash algorithm finalization function that uses the Merkle-Damgård construction [2.5.1]. The compression function is used at least twice but there is only one hash function call. Comparing ENVELOPE with the basic MAC, ENVELOPE adds the cost of the padding of the key, and the additional cost of one or two extra interactions, depending on the key size and the selected CRHF.

The best attack against MD5-Envelope is 2^{96} time with 2^{89} table or 2^{113} time with 2^{66} table, both with a success probability of 87% (Chen & Jin, 2011).

2.6.4 CBC-MAC

CBC-MAC is a technique to construct a MAC from a block cipher. The message is encrypted in CBC mode [2.4.3], where each ciphered block depends on the previous block. Because of such interdependency, the last block cannot be computed without the proper key or if the ciphertext has been tampered during transmission. CBC-MAC uses zeros as the IV. The last block is the authentication tag. The security of CBC-MAC depends solely on the strength of the block cipher algorithm (Bellare, et al., 2000).

2.7 Attacks on CRHF and Encryption Algorithms

Cryptanalysis is the study dedicated to obtain a plaintext for a given ciphertext. However cryptanalysis ultimate goal is to deduce the encryption key used to encrypt a set or plaintexts (Stamp & Low, 2007). An encryption scheme or algorithm is considered broken, when cryptanalysis is feasible. Feasible meaning the effort to break the system costs less than the value of the encryption information or the required time to decipher messages is less than the life span of the information (Lenstra & Verheul, 2001). Concerning a CRHF, the main goal of any attack is either to find collisions or to reduce the complexity to recover the preimage or produce a second preimage, while in the case of MACs, recovering the key (key recovery attack) or producing a second preimage without knowing the key is the aim of the attacks.

The brute force attack is the most inefficient attack. The idea is to try every possible key combination until a successful decryption is achieved. In CRHF, the idea is to try every possible input to find the preimage or a second preimage. If a brute force attack can randomly select the next key to be tested not repeating and previous key, it is expected the brute force attack can succeed after testing half of the key space or 2^{n-1} where n is the key length in bits.

There are numerous attacks on collision resistant hash functions and encryption algorithms. We selected the most relevant to our work.

2.7.1 The Birthday Attack

The Birthday Attack weakens any hash function by exploiting the mathematics behind the birthday problem in probability theory (Jin, et al., 2017) (McKinney, 1966).

The Birthday Problem concerns the probability that, given a set of k randomly people, two of them will have the same birthday. With just 23 people, the probability is 50%, while with 30 people the probability rises to 70%. Applying the birthday problem to hash functions, the attacker wants to find any pair of messages m and m' producing the same hash output, H(m') = H(m). For a hash function of 2^n possible outputs, where n is the fixed length size in bits of the hash output, the probability of finding a collision is

$$p(n) = \sqrt{\frac{\pi}{2}2^n}$$

As a result, the equation lowers the time complexity for obtaining a collision of two random inputs from ***O(2^n)*** to ***O(2^{n/2})***.

2.7.2 The Length Extension Attack

The Length Extension Attack is the concatenation of a second message with the first one, in such a way that the second message produces the same hash output (2nd preimage) in sub exponential or polynomial time. The Merkle-Damgård construction is vulnerable to this attack. The attacker does not need to know the secret part of the first message, for example a prefix or a suffix key, used for authentication. The attacker requires only the length of the secret part (Vũ, 2012).

2.7.3 Known Plaintext and Chosen Plaintexts Attacks

In the known plaintext attack, the adversary, somehow, gained access to several pairs of plaintexts and ciphered texts encrypted with a key. The chosen plaintext attack is more powerful because the adversary has the ability to ask the sender to encrypt plaintexts of his choosing and collect the ciphered text. However the goal is the same for both attacks: to deduce the encryption key and decrypt any messages encrypted with such key (Stamp & Low, 2007). Known plaintexts were fundamental in finding the daily keys of the Enigma Cipher Machine, during World War II. At the time, any known plaintext or suspected plaintext were denoted "cribs". Whenever the Germans broadcasted a continuation of a previous message the plaintext would start with FORT (*Fortsetzung*) plus the time of the first message twice, bracketed by the letter Y. This protocol became known as FORTYWEEPYWEEPY (continuation of the message sent at 2330. Letters also represented numbers). By knowing part of the plaintext, the ciphered text and the index position of known plaintext in the original message, British cryptanalysts could program the Enigma decipher machines (*Bombe*) to stop processing when they found the keys that matched the Forty-Weepy-Weepy protocol (Mahon, 2003-2007). The British also performed the chosen plaintext attack, by mining Atlantic grids which they have not the equivalent German reference. They knew the message about the minefield would be transmitted both using the "*dockyard cipher*" and the Enigma Machines inside the *U-Boats* (Morris, 1993).

2.7.4 Related-Key Attack

In this attack, the adversary knows (or chooses) a relation between several keys and is given access to encryption functions with such related keys, even though she has no knowledge of the keys themselves. The goal of the attacker is to find the encryption keys, by finding a function to compute a possible key given a sample of relations between several keys (Biryukov, 2011).

2.7.5 Differential Cryptanalysis

Differential Cryptanalysis is usually a sophisticated form of chosen plaintext attack applicable to block ciphers, stream ciphers and also hash functions. It studies how slight differences in information input can affect the corresponding output of an encryption algorithm or hash function. If the attack is able to trace differences through the substitution-permutation network or how the input is transformed in each round of a compression function and such traces exhibit a non-pseudo-random behavior it may exploit those non-random properties and recover the secret key or to force collisions in a hash function. It was originally designed to break the FEAL cipher (Stamp & Low, 2007).

2.7.6 Linear cryptanalysis

Linear Cryptanalysis is also a sophisticated chosen plaintext attack applicable to block ciphers, stream ciphers and hash functions. The goal of linear cryptanalysis is to exploit eventual biases in the substitution permutation network. The idea is to approximate the operation of a portion of the cipher with an expression that is linear where the linearity refers to a mod-2 bit-wise operation, like the bitwise XOR, for instance (Heys, 2002).

The attack is divided in two phases: construction linear equations relating plaintexts, ciphered texts and key bits equivalents and deriving key bits from known plaintexts and ciphered texts in conjunction with those equations. When those derivations have diminish the quantity of unknown key bits considerably, a brute force attack becomes feasible and it is performed to recover the encryption key. Like Differential Cryptanalysis it was also utilized to break the FEAL cipher (Stamp & Low, 2007).

2.7.7 Side Channel Attack

Side channel attacks are related to hardware construction details that leak information rather than a weakness of the cipher algorithm or hash function design or implementation, meaning it is neither a design flaw obtained through cryptanalysis nor it is an implementation bug. Side channel attacks can obtain information on plaintexts or keys from electromagnetic leaks, power consumption, CPU usage spikes, timing information and even sounds can be utilized to gain knowledge on encryption keys or plaintexts. Since side channel attacks rely on the relationship between information leaked through a side channel and the secret data, mitigation of such weaknesses fall into two main categories: Either the elimination or the reduction of the emission of such information or to eliminate the relationship between the leaked information and the secret data. One has to make the leaked information unrelated or rather uncorrelated (e.g., by introducing random timing shifts and wait states or by use of dummy instructions) (Chen, et al., 2010) (Zhou & Feng, 2005).

2.7.8 The Encryption Oracle

In order to prove an encryption scheme is secure or insecure, a mathematical theorem assumes the cryptanalyst have access to an encryption oracle, a theoretical machine which always returns a ciphertext μ from a query q produced by an encryption function $E_K(m)$. In other words, the Attacker is capable of performing a chosen plaintext attack and to observe the ciphertexts instantly.

When the Attacker becomes capable of distinguishing patterns between the ciphertexts and random garbage, she gains advantage (Adv) over the encryption scheme (Bellare, et al., 1997).

2.7.8.1 Left-or-Right Indistinguishability

Left-or-right indistinguishability is composed of two games. The attacker is capable of querying in the form of (x_0, x_1), where x_0 and x_1 are messages of equal length. In the first game, each message is responded by encrypting x_0, the left message. In the second, the response comes from x_1, the right message. The formal definition of the left-right-oracle is $E_K(LR(\cdot,\cdot,b))$, where $b \in \{0,1\}$. If $b = 0$ then the oracle computes $C = E_K(x_0)$. Otherwise it computes $C = E_K(x_1)$. *"An encryption scheme is considered "good" if a "reasonably" adversary cannot obtain "significant" advantage in distinguishing cases b = 0 and b = 1 given access to the left-right-oracle"* (Bellare, et al., 2000).

2.8 Post-Quantum Cryptography

The "Post-quantum cryptography" term refers to the ability of a cryptographic algorithm to resist attacks from a quantum computer (Bernstein, 2009). At this time, there is a commercial 20qubits quantum computer from IBM (Lardinois, 2019). N qubits in quantum computers are equivalent to 2^n bits from a classical computer. Besides the large number of equivalent bits that can be processed by a single quantum instruction, other properties from quantum physics (Feynman, 1982) allow certain NP problems from classical computing to be resolved in polynomial time in quantum computers. In the cryptography domain there are two quantum algorithms of the utmost importance: The Shor's algorithm and the Grover's algorithm.

The Shor's algorithm reduces the complexity of the integer factorization problem, the discrete logarithm problem or the elliptic-curve discrete logarithm problem from $\boldsymbol{O(2^n)}$ to $\boldsymbol{O(n^3)}$ (Shor, 1997)**.** As a result, asymmetric cryptography in use today would be useless. We will need algorithms resistant to the Shor's algorithm such as Lattice-based cryptography (de Magalhães, 2014).

In relation to symmetric key cryptography, the Grover's algorithm reduces the effectiveness of an algorithm from $\boldsymbol{O(2^n)}$ to $\boldsymbol{O(2^{n/2})}$ (Grover, 1996), meaning a brute force attack, which we can expect to succeed in finding the encryption key in 2^{n-1} time, will find the key in $2^{n/2}$ time. As a result, we will need encryption algorithms with double the current key length to resist a quantum computer attack.

2.9 Key Management

According to Kerckhoff's principles [2.1], encryption keys should be easy to remember, the users must be able to change keys at any time, but the keys must be strong enough, so that they will not be easily guessed or recovered by an adversary. Such principle is not an easy task to achieve in real life. Keys have a life cycle: They need to be generated, employed, stored during their life cycle and destroyed after their useful life, all this in a very carefully manner.

The NIST have specified 19 different types of keys along with their validity period (Barker, 2016). Dahab & Lopez-Hernandez (Dahab & López, 2007) summarized key management and the 3 main types of keys. In relation to our work, the Dahab and Lopez-Hernandez key definitions suites us better. However, we follow the NIST validity period for keys presented in this section.

2.9.1 Session Keys

Session keys are ephemeral and symmetric used during a single communication session between two endpoints. When the session is terminated, the keys are discarded. They must not be stored. Session keys are used to encrypt and decrypt data transmitted between the endpoints. If the endpoints are using a stream cipher algorithm, the session key is used to generate the stream keys or one-time pads.

Depending on the cryptosystem, session keys may not be used during the entire session. A validity period such as one hour or an amount of transmitted data such as 100MB can be set as limits for a session key. When either of the limits is reached a new session key is negotiated.

2.9.2 Long Term Keys

These are non-ephemeral keys used to generate session keys, keys for digital signature or message authentication (MAC). For symmetric keys, long term keys are usually wrapped (encrypted with a higher hierarchy symmetric key) before they are persisted. They can be also protected by smartcards, dedicated hardware, passwords or passphrases. A key that encrypts another key is called a **Wrapping Key**. According to the NIST a long term key or wrapping key should not exceed 2 years.

2.9.3 Master Keys

Master keys are critical keys, with a very long life span, such as certification authority private keys. Not only their storage requires dedicated hardware but also a fractioned shared secret, so that no single person can access a master key alone. For private asymmetric keys, the life span depends on the key size. For symmetric keys, a master key is never used for encryption or decryption. It is used to derive other keys instead. Its life span should not exceed 1 year.

2.9.4 Passwords and Keys

An encryption key is a secret value independent of the plaintext and of the encryption algorithm used to lock (encrypt) or unlock (decrypt) sensitive information (Stallings, 2011). However, once two endpoints have negotiated a Session Key or a Shared Key, such key can also be used for authentication purposes [2.6]. In order to authenticate a message with a key, one must have the proper key. The authentication is done with what you have.

A password is a secret used for authentication purposes (Brose, 2014). Authentication by password is called Knowledge Based Authentications or authentication with what you know. The user must prove she knows her password to be authenticated. A password can also be defined as a human memorizable key (Bellare, s.d.). A password can also be used as a Master Key.

What you know, what you have and who you are (biometrics) are access control authentication methods (ROSEMBERG, 2014), section 2.2.

2.10 Authentication Protocols

An authentication protocol is a set of rules and operations, which in most cases involves cryptographic algorithms or cryptographic hash functions with the objective of authenticate the endpoints of a communications session. The authentication consists on the verification of the credentials presented by each endpoint to the other to prove the claimed identity (Zuccherato, 2014).

Simple authentication protocols, such CHAP (Challenge Handshake Authentication Protocol) provide authentication only, while more sophisticated ones like TLS (Transport Layer Security) or SRAP (Secure RDF Authentication Protocol), provide authentication and confidentiality by negotiating a session key for a communications session.

Authentication can be done by passwords, asymmetric keys of digital certificates, Personal Identification Numbers (PIN), biometric information or a combination of the previous mentioned methods (Anil, et al., 2004).

There are numerous authentication protocols in use and proposed. In this section we describe the most relevant to our work.

2.10.1 Strong Mutual Authentication with a Shared Symmetric Key

The following protocol, proposed by Stinson authenticates both endpoints Client, C and Server, S, assuming C and S know each other respectively identities (C, S) and a shared symmetric key K_{CS}. As a result they either accept or reject their identities (Dahab & López, 2007). Please note that the communications is started by the client. This step is omitted.

1	S:	random(r_s);
		→C: (S, r_s);

2	C:	random(r_c);
		$y_1 \leftarrow$ MAC(K_{CS}, C‖r_s‖r_c) ;
		→S: (r_c, y_1);

3	S:	$y'_1 \leftarrow$ MAC(K_{CS}, C‖r_s‖r_c); If $y_1 \neq y'_1$ then reject C and Stop; Else $Y_2 \leftarrow$ MAC(K_{CS}, S‖r_c) ; →C: (y_2);
4	C:	$y'_2 \leftarrow$ MAC(K_{CS}, S‖r_c); If $y_2 \neq y'_2$ then reject S and Stop; Else Both Client and Server are authenticated

For this protocol to work, MAC (K, Message) generated by hash algorithm must not be vulnerable to the Length Extension Attack or both endpoints first validate the parameters received, ensuring they are well formed.

2.10.2 Strong Mutual Authentication with Public Keys

Stinson also proposed an authentication protocol using digital certificates and asymmetric keys, similar to the previous one. Both, Client, C and Server, S have digital certificates $CERT_C$, $CERT_S$ which contains their respective identities and public keys P_C and P_S. Their private keys S_C and S_S are not shared. They exchange certificates and digital signatures DS_C and DS_S. If the verifications of each other digital signatures succeed, they are authenticated. Otherwise the authentication is rejected (Dahab & López, 2007).

1	S:	random(r_s);
		→C: ($CERT_S$, r_s);

2	C:	random(r_c);
		$DS_C \leftarrow$ SIGN(S_C, S‖r_s‖r_c);
		→S: ($CERT_C$, DS_C);

3	S:	Validate $CERT_C$; If not verify(P_C, DS_C) then reject C and Stop Else $DS_S \leftarrow$ SIGN(S_S, C‖r_c); →S: (C, DS_S);
4	C:	Validate $CERT_S$; If not verify(P_S, DS_S) then reject S and Stop Else Both Client and Server are authenticated

For this protocol to work, the certificates must be signed by a trusted third party or a Public Key Infrastructure (PKI) must be deployed. Otherwise, a man in the middle attack is can break the protocol.

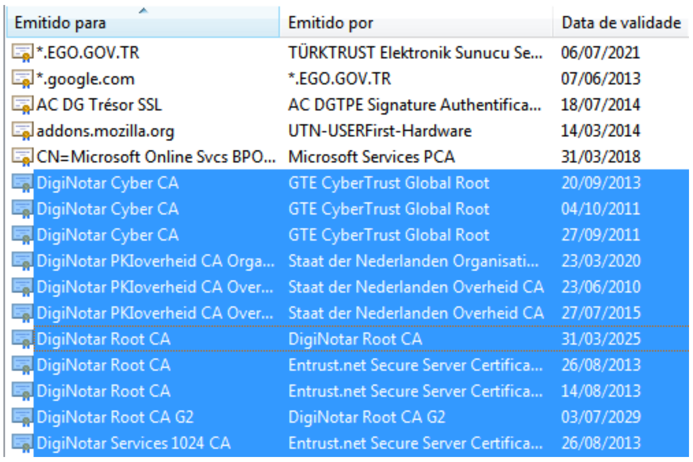
2.10.3 TLS – Transport Layer Security

TLS is the state of the art in authentication and session key establishment. TLS uses digital certificates a PKI infrastructure and a suite of cryptographic algorithms and hashes to enforce authentication, session key negotiation, session confidentiality and message integrity.

The problem with TLS is in the way it is used in real world applications. It is supposed to require digital certificates for both clients and servers, but clients hardly ever use digital certificates, because of financial cost constraints. Also, the large numbers of Certification Authorities (CAs), which are the trusted third party, have become the “Achilles Heel” of TLS.

A clean installation of Windows 7 trusts 13 root CAs. A clean installation of Windows 10 trusts 16 CAs. The latest major update (1.803) at the time of this work shows that Windows 10 trusts 53 root CAs. SSL Observatory claims they observed 1,482 CA certificates trusted by Windows or Firefox, which includes the intermediate CAs certificates. They also observed 651 distinct organizations with authority to sign certificates. However ownership and jurisdiction of those organizations overlap. Those are 2010 numbers (Eckersley & Burns, 2010).

A root CA should sign certificates for an intermediate CA only. However intermediate CAs, depending on the certificate received can sign for a lower hierarchy intermediate CA or an endpoint. The lowest hierarchy intermediate CA should sign only certificates for an endpoint. An endpoint can be a client, a single server (single finality), a domain (multiple finalities) or an entire enterprise. Nevertheless if a root CA signs a certificate for a domain, for example, such domain certificate would be trusted by everyone who trusts such a root CA. The same is valid for any intermediate CA. If a single private key from hundreds or a maybe few thousand CAs is compromised, it can sign false digital certificates for any social network, government organization, financial institution, major e-commerce sites and others, provoking chaos in web transactions until the certificate whose private key has been compromised is revoked. This has happened at least once with DigiNotar in 2011 in a period of 7 months (Schwartz, 2011). DigiNotar root CA certificate had a valid date up to 2025. If not discovered, one cannot even estimate the damage it could have caused.



Emitido para	Emitido por	Data de validade
*.EGO.GOV.TR	TÜRKTRUST Elektronik Sunucu Se...	06/07/2021
*.google.com	*.EGO.GOV.TR	07/06/2013
AC DG Trésor SSL	AC DGTPE Signature Authentifica...	18/07/2014
addons.mozilla.org	UTN-USERFirst-Hardware	14/03/2014
CN=Microsoft Online Svcs BPO...	Microsoft Services PCA	31/03/2018
DigiNotar Cyber CA	GTE CyberTrust Global Root	20/09/2013
DigiNotar Cyber CA	GTE CyberTrust Global Root	04/10/2011
DigiNotar Cyber CA	GTE CyberTrust Global Root	27/09/2011
DigiNotar PKIoverheid CA Orga...	Staat der Nederlanden Organisati...	23/03/2020
DigiNotar PKIoverheid CA Over...	Staat der Nederlanden Overheid CA	23/06/2010
DigiNotar PKIoverheid CA Over...	Staat der Nederlanden Overheid CA	27/07/2015
DigiNotar Root CA	DigiNotar Root CA	31/03/2025
DigiNotar Root CA	Entrust.net Secure Server Certifica...	26/08/2013
DigiNotar Root CA	Entrust.net Secure Server Certifica...	14/08/2013
DigiNotar Root CA G2	DigiNotar Root CA G2	03/07/2029
DigiNotar Services 1024 CA	Entrust.net Secure Server Certifica...	26/08/2013

Figure 19 - DigiNotar Revoked Certificates from Windows 7

Jacob Appelbaum, a core member of the Tor Project stated: *"We cannot determine whether they succeeded in creating any intermediate CA certs. That's really saying something about the amount of damage a single compromised CA might inflict with poor security practices and regular internet luck"*

Further details on TLS and how it can be exploited can be found in (ROSEMBERG, 2014) pp. 46-50.

The TLS handshake protocol is described below:

`ClientHello`	$C \to S$	$C,\ Ver_C,\ Suite_C,\ N_C$
`ServerHello`	$S \to C$	$Ver_S,\ Suite_S,\ N_S,\ \text{sign}_{CA}\{S, K_S^+\}$
`ClientVerify`	$C \to S$	$\text{sign}_{CA}\{C, V_C\},$ $\{Ver_C, Secret_C\}_{K_S^+},$ $\text{sign}_C\{\text{Hash}(\text{Master}(N_C,\ N_S,\ Secret_C) + Pad_2 + \text{Hash}(Messages + C + \text{Master}(N_C,\ N_S,\ Secret_C) + Pad_1))\}$
⟨Change to negotiated cipher⟩		
`ServerFinished`	$S \to C$	$\{\text{Hash}(\text{Master}(N_C,\ N_S,\ Secret_C) + Pad_2 + \text{Hash}(Messages + S + \text{Master}(N_C,\ N_S,\ Secret_C) + Pad_1))\}_{\text{Master}(N_C,\ N_S,\ Secret_C)}$
`ClientFinished`	$C \to S$	$\{\text{Hash}(\text{Master}(N_C,\ N_S,\ Secret_C) + Pad_2 + \text{Hash}(Messages + C + \text{Master}(N_C,\ N_S,\ Secret_C) + Pad_1))\}_{\text{Master}(N_C,\ N_S,\ Secret_C)}$

Figure 20 - TLS Handshake Protocol (Mitchell, et al., 1998)

2.10.4 SRAP – Secure RDF Authentication protocol

In our Master's dissertation, we proposed SRAP as an alternative to TLS with the primary goal of reducing the dependency on CAs. SRAP uses self-signed digital certificates for both clients and servers (ROSEMBERG, 2014).

SRAP takes advantages of Web Semantic concepts such as RDF, OWL and Linked Data to construct a social graph where servers can vouch for the authenticity of other servers, building a Web of Trust. A copy of the client's public key exists in the client's RDFK file (hidden RDF) located in the client's personal web server.

Servers that vouch for other servers are called authentication partners (APs). If the server whom the client is trying to authenticate for the first time do not trust any of the APs, the authentication partner of last resort APRL must be used to authenticate the server. The APRL is the only one who must have a digital certificate signed by a CA.

The trust, in the client's context, exists if the client has already authenticated itself with at least one of the APs. From the server's context, trust in client's identity exists if any of the APs can vouch for the client's identity, meaning the client has authenticated with at least one of the APs and at least one of the APs has a copy of the client's RDFK file in its identity repository. If the client has not yet authenticated itself with any of the APs, it informs the server the location of its RDFK file in its personal web server.

Both the server's identity and the client's identity are verified by a challenge. The challenge consists of encrypting a random string with the public key of the challenged endpoint and sending both the encrypted string and a hash of the random string to the challenged endpoint. The challenged endpoint will use its private key to decrypt the encrypted random string, calculate its hash, comparing it with the received hash. If they match, the second endpoint returns a new hash (different hash algorithm) of the random string. The payload returned to the challenging endpoint is only the new hash. The challenging endpoint will calculate the new hash of the random string and compare it with the received second hash. If they match, the challenged endpoint is authenticated and the random string can be used as a session key.

Once both client and server have been authenticated, they store each other’s RDFKs in their identity repository. As a result, second and future authentications can be done much faster by a modification of Stinson’s mutual authentication protocol with digital certificates. A trusted third party is not needed, since the endpoints’ public keys have already been authenticated in the first time they authenticated each other. SRAP simply challenge both endpoints. We called this protocol SRAP Fast Authentication. However, if both endpoints negotiate a long term symmetric key in the first authentication, second and future authentications may be achieved even faster without the need to use computationally expensive asymmetric key operations. When the lifetime of the long term symmetric key expires, the endpoints can negotiate another one with the use of SRAP Fast Authentication.

2.10.4.1 SRAP Disadvantages

Although an attacker has fewer opportunities to disrupt SRAP than she has to disrupt TLS, if SRAP is exploited it could take a long time for the endpoints to notice they were attacked. For example, if the attacker is successful in replacing the client’s RDFK file in the client’s personal web server, and the targeted server has not yet authenticated the targeted client before, the attacker can deceive the targeted server and then restore the client’s original RDFK file. The targeted server will trust a false client until the real client tries to authenticate itself with the targeted server. The credentials presented by the real client will not match the ones already in possession by the server.

Suppose the attacker is able to gain access to the private key of an AP of the targeted server and the AP is already trusted by the targeted client. If the attacker is able to replace the targeted client RDFK file by a false one, she can successfully perpetrate a man-in-the-middle attack. After the first authentication with the targeted server, she can return the original RDFK file of the targeted client. The attacker will be able to intercept and forge communications between the targeted client and server, as long as she can maintain herself between the attacked endpoints. The targeted client will be able to detect the attack only if it tries to authenticate itself with the targeted server from a device, which the attacker cannot position herself between the new device and the server.

The full description of SRAP protocols can also be found at (ROSEMBERG, 2014), section 6.

3 Hypothesis and Experiment Design

In this section, we report the hypothesis formulated to refute or confirm the research questions and the methods used to assess the experiments.

3.1 Hypothesis for the research questions

RQ_1: Are software developers familiarized with the basic concepts of symmetric key cryptography?

H_{01}: Developers have clear knowledge of the basic concepts of symmetric key cryptography.

H_{A1}: Developers confuse basic concepts of symmetric key cryptography such as block sizes and key sizes. They also do not know the key is secret and must be negotiated in a secure channel. They do not know the purpose of the IV and why a padding scheme is necessary for block ciphers. As a consequence they are bound to violate cryptographic rules.

RQ_2: Are developers capable of consciously choosing a safe symmetric key encryption algorithm?

H_{02}: Given an API, the developer chooses the AES or the Twofish encryption algorithm with any encryption mode other than ECB. The developer uses a random IV and she is aware of her choice.

H_{A2}: The developer chooses an unsafe encryption algorithm, an unsafe mode of operation or she chooses a safe algorithm and mode of operation by chance, not being sure whether the selection is safe.

RQ_3: Can developers use the AES (CTR Mode) encryption algorithm correctly?

H_{03}: The developer uses a random and unique IV for each encryption that uses the same encryption key, as a result, the same plaintext encrypted with the same key several times will produce different ciphertext each time.

H_{A3}: The developer uses a constant IV.

RQ_4: is it possible to use a CRHF in combination with a secret key to generate one-time pads that will not repeat itself for a long time?

H_{04}: Given an unbroken CRHF, a safe MAC Algorithm, a constant encryption key, a counter and one salt per counter, the keystream repeats itself during a session.

H_{A4}: Given an unbroken CRHF, a safe MAC Algorithm, a constant encryption key, a counter and one salt per counter, the keystream does not repeat itself during a session.

RQ_5: Is HX is more effective than other encryption algorithms?

H_{05}: The developer violates as many cryptographic rules using HX as she does using other algorithms or more.

H_{A5}: The developer violates less cryptographic rules using HX than she does using other algorithms.

RQ_6: Is HX is more efficient than other encryption algorithms?

H_{06}: The developer takes more time to write a code that encrypts and decrypts a message using HX than she does using other algorithms.

H_{A6}: The developer takes less time to write a code that encrypts and decrypts a message using HX than she does using other algorithms.

3.2 Experiment Methodology

In order to reject H_{01}, we designed a survey to test the knowledge of the developers in basic concepts of symmetric cryptography with emphasis on block cipher algorithms, taking into account that AES and Twofish are block cipher algorithms, they are the widely available algorithms and AES algorithm is the state the art in symmetric key cryptography. Initially we expected the survey was also enough to reject H_{02} and H_{03}. However the survey was not the best method to accomplish the rejection of those null hypotheses.

To reject H_{04}, we need to test if the use of an encryption key, a counter and a salt used as parameters for a safe MAC Algorithm will generate pseudorandom keystreams that will not repeat itself for the duration of a session. We selected RIPEMD160, SHA-256, SHA-384, SHA-512 and Whirlpool as CRHFs and HMAC and ENVELOPE algorithms as keystream generators, where the message was the counter concatenated with the salt. If a billion unique keystreams can be generated for each CRHF and Keystream generator algorithm pair, H_{04} is rejected.

For H_{02}, H_{03}, H_{05} and H_{06} rejection, we designed a controlled experiment empirical study. Each participant had to implement three tasks to encrypt and decrypt a very important message five times in a row. Each task had to be timed.

In the first task, the developer had to choose an encryption algorithm from her choice. In the feedback form, she had to justify why she selected the chosen algorithm to prove she was aware of her choice.

In the second task, she had to implement a code similar to the code of the first task, but she had to use AES-CTR mode. If she chose AES-CTR mode in task one, she had to use AES-CBC as an alternative.

In the third task, the code had to be implemented using the HX algorithm.

Half of the participants started with task one, then task two and finished with task three while the other half started with task three first, then task one and finally task2. This division helps to mitigate the influence on the others tasks due to the possibility of fatigue and frustration for not been able to complete one of the tasks.

At the end of the experiment, developers were asked to fill the feedback form, where they had to justify their algorithm's choice for the first task, among other questions.

3.3 Survey Design

As stated in the background section [2.4.4.7], the AES block cipher algorithm is one of the most secure block cipher encryption algorithms and widely available in most APIs and even in most modern CPU instructions sets. The goal of the study is to verify the familiarity of software developers regarding the basic concepts of symmetric cryptography, such as key management, IV randomness and Padding. These concepts are necessary to correctly use the AES algorithm.

3.3.1 Target Audience, Population and Sample

The target audience of the study was software developers with at least one year of experience in software development. We believed experienced developers, preferably with previous experience in the use of cryptographic algorithms would know relatively well the basic concepts of cryptography. However, only a participant who either has not enough experience in software development or had never used a cryptographic algorithm was classified as an outlier. The survey was designed in Portuguese (the native language of most of the participants and required language for foreign students). In this sense, we selected graduate students of the Informatics course of PUC-Rio to compose the survey population. All of the participants had to sign the consent form [Annex 1].

3.3.2 Subjects Characterization

In order to filter possible outliers, we applied a characterization questionnaire composed of four filter questions (LinÂker, et al., 2015).

1. The highest academic degree (High School, Bachelor, Masters, Doctorate)
2. How many years of experience as a software developer
3. In how many projects has the participant used cryptographic algorithms
4. How well familiarized the participant was with symmetric key encryption theory. This question was designed on a Likert Scale (Allen & Seaman C, 2007) with the following possible choices: (Not

familiarized at all; Not well Familiarized. I feel unsafe to use them; I'm fairly familiarized. I have no problems in using them, if I have access to the documentation; Well Familiarized; Totally Familiarized)

3.3.3 Substantive Questions

We designed seven substantive questions (5 to 11) (LinÂker, et al., 2015) (Kasunic, 2005) to help us reject H_{01}. All of the substantive questions are closed-ended questions (multiple choices). Questions 5 and 11 also asked the participants to justify the answers. We asked the participants to answer the questions in sequence and not to turn the page until all of the questions on each page had been answered to mitigate the influence one question could have on the others, especially the last question, which gives information about Padding schemes.

5. Which cryptographic algorithm would you choose to encrypt a text file, containing confidential information?

The possible choices were:

a) DES
b) 3DES
c) AES ECB 256
d) AES CBC 128
e) RC4
f) Blowfish

6. When you read SERPENT 256, for you, what is the meaning of the number 256?

The possible choices were:

a) Key size in Bytes
b) Key size in bits
c) Algorithm's block size in Bytes
d) Algorithm's block size in bits
e) Both key size and Algorithm's block size in bits

7. The CBC (Cipher Block Chaining) mode may use an IV (Initialization Vector [I]). For you, the IV is required or optional and what is its purpose?

The possible choices were:

a) Optional. Strengthening of the cryptographic algorithm
b) Required. Fill the message with the necessary bits to complete the block size
c) Optional. Mislead the Cryptanalyst by putting random information which will not be used in the encryption
d) Optional. If not supplied by the developer, the algorithm will automatically create a random IV
e) Optional. If not supplied by the developer, the algorithm will use a default value for the IV

8. In your understanding, what is the purpose of Padding in block cipher algorithms?

The possible choices were:

a) Strengthening of the cryptographic algorithm
b) Fill the message with the necessary bits to complete the block size
c) Mislead the Cryptanalyst by putting a random information which will not be used in the encryption
d) Optional. If not supplied by the developer, the algorithm will automatically create a random Padding
e) Optional. If not supplied by the developer, the algorithm will use a default value for the Padding

[I] The IV is required in CBC mode [2.4.3]

9. In relation to cryptographic keys and block cipher algorithms, what is your understanding about the key size?

The possible choices were:

a) The greater the key size, the better
b) The key size must be exactly the same size of the block of the algorithm
c) The key size must be exactly the size in bits specified by the algorithm, no matter the block size
d) If the key size is greater than the algorithm’s block size, the algorithm will truncate the key to the block size automatically
e) If the key size is less than the algorithm’s block size, the algorithm will automatically concatenate the key with the necessary blanks (spaces) needed to match the block size

10. Concerning cryptographic Keys and IVs, what must be secret and what must be transmitted openly with the ciphertext?

The possible choices were:

a) Both must be secret. The Key and the IV must be negotiated in a secure channel
b) Both must be transmitted openly with the ciphertext
c) The Key is secret and must be negotiated in a secure channel. The IV must be transmitted openly with the ciphertext
d) The IV is secret and must be negotiated in a secure channel. The Key must be transmitted openly with the ciphertext
e) Both must be secret. The HASH of the IV with the key must be informed

11. Concerning Padding Schemes observe the following explanation about Padding and choose the safest scheme in your opinion. Also, please explain why you chose your preferred Padding Scheme.

```
ANSI X.923 ...|DD DD DD DD DD DD DD DD | DD DD DD DD 00 00 00 04|
```

`In ANSI X.923 Bytes filled with nulls (00) are padded and the last Byte defines the padding length including the last Byte.`

```
ISO 10126 ... |DD DD DD DD DD DD DD DD | DD DD DD DD 81 A6 23 04|
```

`ISO 10126 Bytes filled with random Bytes are padded and the last Byte defines the padding length including the last Byte.`

```
PKCS7 (extension of PKCS5)

01
02 02
03 03 03
04 04 04 04
05 05 05 05 05
06 06 06 06 06 06
```

`The value of each added Byte is the number of bytes that needed.`

```
Padding with Nulls
... |DD DD DD DD DD DD DD DD | DD DD DD DD 00 00 00 00|
```

```
Padding with Blanks
... |DD DD DD DD DD DD DD DD | DD DD DD DD 20 20 20 20|
```

The possible choices were:

a) I prefer ANSI X.923 because of its simplicity and confusion avoidance between valid nulls (nulls which are part of the original message) and invalid nulls (the ones that are padded to the message)
b) I prefer ISO 10126 because random bits will make it more difficult for a cryptanalyst

c) I prefer PKCS7 because it is even more guaranteed to identify valid and invalid bits (padded bits)
d) I prefer filling the message with Nulls or Blanks because there is nothing simpler than these schemes
e) I prefer a scheme that exists on any API in order to guarantee interoperability

3.3.4 Confounding Factors and Threats to Validity

If a participant chooses a correct answer it does not mean she really knows the correct answer. She might have guessed.

We cannot control the level of commitment of the participants with the experiment. The lack of commitment poses a threat to the validity of the experiment, since the participants might have chosen to answer the questions of the survey randomly.

3.4 Experiment to Reject H_{04}

For each CRHF and MAC algorithm used as a keystream generator, we generated one billion keystreams. A constant key was used in the experiment. Each time a new keystream was generated it was compared with all the other previously generated keystreams. If the last keystream is unique, H_{04} can be rejected. The counter (i) started at 0 and was incremented by one for each keystream generated. At the same time we generated the salt (S_i) with random values ranging from 0 to 2^{31}. The keystream generator formulas are explained in [4.1] and [4.3].

3.5 Controlled Experiment Empirical Study

In this study, we were interested in testing the maturity level of developers in relation to encryption algorithms. Particularly, we wanted to verify if they could select a safe encryption algorithm from the available set of the Cipher class of the Java Language and if they were able implement a code that encrypted and decrypted messages without violating the rules we selected in section [1.4] pp.18.

It was also our intention to compare the effectiveness and the efficiency of AES, HX and any other algorithm chosen by the participants. The effectiveness and the efficiency of both algorithms were our usability metrics.

3.5.1 GQM

According to Claes (Claes, et al., 2000), a goal-definition template, which identifies the objects, goals, quality focus and the perspective of the study, ensures that important aspects of an experiment are defined before the planning and execution of an empirical study. In this study, we used the Goal Question Metric model (Basili & Weiss, 1984) for the elaboration of evaluation plans of usability of encryption algorithms.

GQM Questionnaire	
Analyze	HX and AES algorithms
With the purpose of	Evaluate the effectiveness and efficiency of HX, when compared to AES
Focusing	The identification of insecure code and the usability of encryption algorithms
In relation to	Sensitive information protection
From the point of view of	Software Developers
In the context of	Symmetric Key Cryptography usability

Table 5 - GQM Template of the Empirical Study

3.5.2 Confounding Factors and Threats to Validity

1) Formal training in cryptography. Developers who have had training are not expected to violate encryption rules.
2) The Developers' previous experience: Experienced developers might have had their code inspected by a security expert and learned how to avoid the violations of cryptography rules.
3) The Developer's affinity to the Java Language. Since the experiment was conducted in Java, Developers who were not familiarized with Java may have produced a worse code than the ones who were familiarized.

4) Poor code examples. Developers were free to research on the Internet and to copy and paste code from any forum they chose. The quality of those codes might have influenced the results.
5) The fatigue and frustration for not being able to complete one of the tasks might influence on the outcome of the others.

3.5.3 Target Audience Identification

As target audience, we invited graduate students of the Department of Informatics of PUC-Rio or professional developers from software houses.

3.5.4 Participants Characterization

In order to identify possible outliers, we asked each participant to fill the characterization form. We wanted to know the academic level of each participant, how much experience they had in software development, how familiar they were with the Java language, who was responsible for inspecting the code they produce for security vulnerabilities (in their organization), if they had any formal training in cryptography, network security or information security. Finally, what was their opinion on who should be responsible in detecting or correcting any encryption rule violation: the developer, the project manager, a security expert, an external plugin or the class itself used for encryption. The participant's characterization form can be found in [Annex2].

3.5.5 Participants Training and Leveling

There was no training or leveling. Since there were no example codes for the HX algorithm, developers received the UML documentation and the Javadoc of the class. No examples or hints in cryptography were provided to the participants. The Javadoc, however, explained the cipher modes available for HX and the hash algorithms available to generate keystreams. It also recommended the developers to set the masterKey property first or they would get an empty string from the encrypt() or decrypt() methods.

The participants received a recommendation to import javax.xml.bind.DatatypeConverter and to use the method DatatypeConverter.printHexBinary() to print the encrypted message.

The participants were told that in order to use the HX class, they had to import the puc.galgos.crypto.HX package.

3.5.6 Experiment Tasks

The experiment encompassed three tasks. All of the tasks consisted in building a code to encrypt and to decrypt a message 5 times in a row. Each time, the encrypted message and the decrypted message should be printed at the console output. Nevertheless, in the first task, the developer was asked to select an encryption algorithm of her choice. In the second task, the AES-CTR algorithm and encryption mode was mandatory. If the developer chose AES-CTR for the first experiment, she was asked to use AES-CBC instead. Finally in the third task, the developer was asked to use HX. The order of the tasks (1-2-3 or 3-1-2) was randomized for each participant, to mitigate the influence of order of the tasks in the experiment. When necessary, we intervened and directly assigned the task order to balance the order of the tasks distribution.

For each task, the participant should spend at most thirty minutes. Preferably the entire experiment should not exceed one hour per participant. After thirty minutes, the participant should advance to the next task, but we did not interfere with the participant’s choice to finish the current task or give up and move to the next. The intent was to avoid some emotional response or frustration from the part of the subject, which could interfere with the rest of the experiment.

We also instructed the participants to write down the start and finish times of each task, so we could measure the time spent on each task.

If the participants did not violate any encryption rule, other than a poor choice of a key, the encrypted message should be different for each of the five encryptions. A poor choice of a key did not affect the outcome, but was marked as a violation.

In this experiment there were 4 possible encryption rules that could be violated:

1) The use of a safe encryption algorithm

2) The use a safe mode of operation

3) The use of a strong encryption key

4) The use of a random and unique IV

In this experiment, the number of rules violated indicated the effectiveness of the algorithm in the context of the usability of the algorithms. The time spent on each task indicated the efficiency of the algorithm. An algorithm that takes less time to implement a code which produces the same result is more efficient than another, which takes more time to accomplish the same goal. The experiment tasks (HX as the last task) can be found in [Annex3].

3.5.7 Feedback Form

On the last phase of the experiment, each participant was asked to fill the feedback form to provide additional data to our research. In relation to the first task we asked which algorithm the participant chose and why. **We needed to know the justification of the choice to assess if the participant consciously chose a safe encryption algorithm**. We also asked if the participant copied and pasted code from a forum or documentation example. If they did, we asked if they checked the code for any vulnerability that could weaken the encryption. Finally we wanted to have a feedback on the difficulty level of the task. We decided to measure the difficulty level, which is directly influenced by usability of the class, with a Likert scale with the following choices:

1. Very easy. I did not have any difficulty.
2. Easy, even though I had some difficulty, easily overcame.
3. Complicated. I experienced some difficulty and I needed some effort to overcome.

4. Difficult. The task was hard and I needed considerable effort to overcome.
5. Too difficult. I was unable to complete the task

Concerning the second task, the first question we asked if the participants knew the reason why the researchers forced the use of the AES-CTR algorithm. The answers might have provided additional information to confirm the choice of the algorithms made in the first task by the participants.

Like in the first task, we asked if the participant copied and pasted code from forums or on-line documentation, if they checked the code for any vulnerability and how they classified the difficulty level of the task.

We asked a specific question about AES-CTR: If the participants found strange the fact the encrypted message was different each time it was encrypted and gave them five choices for best answer:

1. Yes. I don't know why.
2. No. It is supposed to be this way.
3. With my code, it did not happen.
4. That happened with my code, but I changed it to avoid it.
5. I did not notice and it does not matter. What matters is to encrypt and decrypt the message correctly.

Concerning the third task, we were particularly interest in the usability of HX. In consequence, we asked if the participants opted to change the default attributes of HX and why, the difficulty level of the task, what was their impression about the JAVADOC of the class and what was their impression about the set of methods of HX class. The last two questions were also measured by the Likert scale. In relation to the JAVADOC the choices were:

1. Very poor. Insufficient information.
2. Poor. Lacks example codes.
3. Acceptable. I was able to understand, but it lacks further technical information.
4. Good. It's what is expected from a JAVADOC.

5. Very good. It does not need any other information

About the set of methods of HX, the choices were:

1. Unsuited to the goals of the class.
2. Partially unsuited to the goals of the class
3. Undecided. I can't evaluate.
4. Suited to the goals of the class
5. Well suited to the goals of the class

We also decided to ask a few questions about the experiment. We asked which encryption class the participants preferred Cipher or HX and why, assuming both of them was safe to use. The participants were also free to write about their impression of the tasks, their impression of the experiment forms and to give any suggestion they wanted to the researches. The feedback form (HX as the last task) as well as the participants’ answers can be found in [Annex4].

4 The HX proposal

In this section, we discuss our proposal of a stream cipher encryption algorithm based on CRHF. We also show how SRAP can use a CRHF to authenticate both the client and the server using a long term shared key.

4.1 Formal Description

The encryption processes consists of the following steps:

- The Sender and Receiver negotiate a CRHF (H)

- Generate and distribute a shared key K between the sender and the receiver. The key can be of any length, but the optimal Key size is the size of the digest output of H.

- Divide the message M in n blocks of size |H|, such that $M = m_0 \| m_1 \| \ldots \| m_{n-1}$. For compatibility among operating systems regional code pages, M must be converted to CP-1252 encoding.

- i is the block counter, varying from 0 to n-1.

- For each block, generate a 32 bit integer random number (S_i). S_i must be a DWORD little endian format. The same format is required for the counter i.

- Generate the ciphered block, using HMAC, with the following formula:

$$c_i = S_i \| m_i \oplus H((K \oplus opad) \| H((K \oplus ipad) \| S_i \| i))$$

- Calculate the MAC (Message Authentication Code) = $H(K \| M \| SMAC)$ and prepend it to the ciphertext (only if authenticated encryption is required). SMAC is the salt used for the message authentication code.

As a result, the ciphertext is $C = SMAC \| MAC \| c_0 \| c_1 \| \ldots \| c_{n-1}$

The decryption process is slightly different from the encryption:

- Separate both SMAC and MAC from the ciphertext.

- The ciphered block length is l = |H| + 32 bits long, except for the last ciphered block. The last block length may vary from 40 bits (last salt + one character) to l. For each ciphered block c_i of l bits, take the first 32 bits which corresponds to the block salt (S_i). The remaining l-32 bits is the encrypted message block (Φ_i).

- The original message block is obtained from the formula

$$m_i = \Phi_i \oplus H((K \oplus opad) \| H((K \oplus ipad) \| S_i \| i))$$

- Reassemble the message $M = m_0 \| m_1 \| \ldots \| m_{n-1}$

- Calculate the message's MAC' = $H(K \| M \| SMAC)$

- Check if the received MAC matches MAC'. If they do, the message is both authentic and intact.

4.2 Analysis of the Encryption Scheme

"The construction of a pseudorandom generator from a one-way function provides a solution for symmetric encryption starting from a one-way function" (Bellare, et al., 2000) (Håstad, et al., 1999).

Using the counter and having one Salt for each block generates a unique keystream, enforcing the same message, encrypted with the same key twice, will not produce the same ciphertext, because the salts would be different for the same counter. It emulates the Vernam Cipher, the same way other stream ciphers do, generating a keystream as large as the plaintext and then performing a bitwise XOR between the plaintext and the keystream.

A successful Known-plaintext attack on a specific block allows an attacker to decrypt only that specific block. It reveals the keystream of that block only. It does not reveal the encryption key.

The salts replace the IV in HX. No developer can use a constant IV with HX, since the salts are randomly created during encryption.

A sloppy developer which uses a constant encryption key is less affected in HX than in other algorithms. The same key may be reused to encrypt other messages, since the sequence of Salts generated for each block will be different. A keystream generated with the same key, counter and salt is expected to be repeated with a 50% probability after 2^{16} encryptions (birthday attack on a 32 bit integer). In this case, half of the keystream will be repeated. If HX is implemented with a 64 bit salt, half of the keystream will be repeated after 2^{32} encryptions with the same key.

Because the ciphered block is composed of the salt block and the encrypted block, there is no need for synchronization between the sender and the receiver. However, there is an increase in the size of the ciphertext. For each plaintext block, we get a ciphertext with 32 extra bits. As an example, for a plaintext message of 2GB, the ciphertext increases, according to the following table:

Hash Length (bits)	Hash Length (Bytes)	Number of Blocks	Total Salt Cost (MB)	Final Size (GB)	Increase
160	20	107.374.183	409,6	2,40	20%
256	32	67.108.865	256	2,25	12,5%
384	48	44.739.243	170,7	2,17	8%
512	64	33.554.432	128	2,13	6%
1024	128	16.777.216	64	2,06	3%

Table 6 - Salt Cost

From Table 6, we can infer that the larger the length of the digest output if the CRHF, the stronger the cipher and the lower the salt cost.

The cipher strength depends on the strength of the CRHF, HMAC and the counter mode of operation. Because the encryption algorithm applies the hash to the encryption key, the salt and the counter, the compression function of the hash used up to four times (four if the key size is greater than |H|), strengthening the resistance to second preimage attacks to a complexity nearing ***$O(2^n)$*** (Kelsey & Schneier, 2005). The cost to succeed in a key recovery attack is the cost of the key recovery attack of HMAC with the CRHF, which is infeasible even for the broken MD5 [2.6.2].

Since HMAC is a pseudorandom function, we can expect the keystream to repeat itself close to $2^{n/2}$ blocks (birthday paradox on random distribution). As a result, given an encryption key, the maximum theoretical plaintext size that can be encrypted with HX is $n \cdot 2^{n/2}$ bits. However, with a 31 bit counter used in the implemented version, the maximum plaintext size is $n \cdot 2^{31}$ bits.

The proposed scheme uses $\mathrm{HMAC}_{(\mathrm{CRHF})}$ as a PRF (pseudo-random function) and the counter mode of operation to encrypt messages. Proposition 8 of (Bellare, et al., 1997) proves the counter mode of operation is secure. The authors demonstrate the encryption function is indistinguishable from a PRP (pseudo-random permutation) and that any secure PRP can be converted to a secure PRF.

Proof: Let t be the running time a an oracle query is answered, q the number of queries made to the oracle, μ the number of ciphertext bits returned by the oracle, l the input bits of the PRF function, L the output bits of the PRF function and Adv the advantage when distinguishing a function from random. For a PRP, $l = L$. Then:

$$\mathrm{Adv[PRF]}(t, q) \leq \mathrm{Adv[PRP]}(t, q) + q^2 \cdot 2^{-l-1}$$

Assuming the PRP is indistinguishable from random, $\mathrm{Adv[PRP]}(t, q) = 0$. As a result, if $\varepsilon = \mathrm{Adv[PRF]}$ then $q = \varepsilon^{1/2} \cdot 2^{l/2}$, meaning unless about $2^{l/2}$ keystreams blocks are generated, the adversary's advantage is limited to ε (Xian & Tingthanathikul, 2004).

Theorem 13 [Security of a XORC using a pseudorandom function] (Bellare, et al., 1997) proves any stream cipher using a pseudorandom function is secure if the PRF function is secure. The authors shows that there is a constant $c > 0$ which satisfies the following: for given a function *F(t', q', ε')-secure* PRF family with input length l bits and output length L bits, then, for any q the XORC(F) scheme is *(t, q, μ; ε)-secure* in the left-or-right sense, for $\mu = \min(q'L, L2^l)$ and $t = t' - c \cdot (\mu / L) \cdot (l + L)$ and $\varepsilon = 2\varepsilon'$, where μ is number of ciphertexts returned from the encryption oracle from the queries q. XORC refers to an encryption scheme which uses bitwise XOR operation between the plaintext and the keystream produced by an encryption function in counter mode of operation.

From the above, considering n is the digest output length of a CRHF, if a CRHF is secure, $HMAC_{CRHF}$ is a secure PRF up to $2^{n/2}$ digests produced with the same key (Bellare, 2006) (Kim, et al., 2006) and the counter mode of operation is secure, up to $2^{n/2}$ keystreams segments for any PRF (Bellare, et al., 1997), then we can deduce HX is also secure up to $2^{n/2}$ keystreams.

In order to break HX, an attacker must either break the counter mode of operation, the HMAC algorithm or the CRHF. Since it is infeasible to break or weaken the CTR-mode or HMAC, the best choice for the attacker is to try to break or to significantly weaken the CRHF.

For authentication and integrity, we can use the MAC algorithm because both the key and the plaintext are unknown to the attacker. A Length Extension Attack [2.7.2] with MAC requires knowledge of the plaintext and the key size.

Recently, there have been significant improvements on Hash functions. In (Su, et al., 2016), the authors claim that a non-iterative hash function for small messages can produce a hash output complexity of $\boldsymbol{O(2^m)}$, where $80 \leq m \leq 232$ and $80 \leq m \leq n \leq 4096$ and n being the size of the message to be hashed. Assuming a perfect distribution is achieved such hash algorithm would be immune to the Birthday Attack. Without iteration a construction structure is not needed. As a result the basic MAC algorithm can be applied without the risk of the Length Extension Attack. By using this hash algorithm, the maximum key length would be 3.832 bits (4096 – 232 bits of the counter – 32 bits of the salt). The maximum plaintext size that could be encrypted would be $232 \cdot 2^{232}$ bits.

Skein [2.5.8] and SHA-3 [2.5.7] are immune to the Length Extension Attack, since they do not make use of the Merkle-Damgård structure. As a result the simple MAC instead of HMAC could be applied by HX with those CRHF with significant performance gains.

All of these hash functions can be added to the CRHF set of HX, giving more encryption options to the developers, without the need of changing the HX algorithm itself. However, once a CRHF security is compromised, developers can switch the compromised CRHF for another one of the set with minimal changes in their code.

4.3 Options for HX

We call the use of multiple salts, one per block, the Counter Mode (CTR) of operation for HX. We call the combination of HMAC with CTR operation mode to generate keystreams the HMAC/CTR encryption mode or cipher mode. However HMAC is not the only safe MAC algorithm. The ENVELOPE [2.6.3] technique is also safe to use. ENVELOPE can be twice as fast as HMAC with a marginally smaller security level [2.6.3]. The formal representation of HX in ENVELOPE/CTR encryption mode is:

$c_i = S_i \,\|\, m_i \oplus H(K \,\|\, \pi_K \,\|\, S_i \,\|\, i \,\|\, K)$, to encrypt and

$m_i = \Phi_i \oplus H(K \,\|\, \pi_K \,\|\, S_i \,\|\, i \,\|\, K)$, to decrypt

It is also possible to use an IV instead of multiple salts. We call this mode of operation, the Segmented Integer Counter (SIC). A 32 bit integer is randomly generated automatically by the encryption function. It is converted into a 64 bit integer and left shifted 32 bits. The random value is used as the higher 32 bits of a 64 bit integer. The remaining lower 32 bits are incremented each time a new keystream is generated. The implementation cannot allow the lower 32 bits overflow and increment the higher 32 bits, in order to avoid keystream overlapping. The IV is prepended to the ciphered text.

This mode of operation has an inferior security level, when compared with the original CTR mode but does not have the salt increase cost. In this mode, using the same key, after 2^{16} encryptions there is 50% probability the entire keystream will be repeated, allowing the cryptanalyst to decrypt the messages where the keystream is repeated. With the original CTR mode, only half of the salts are expected to be repeated with the same counters, allowing the decryption of half the message by the cryptanalyst.

The formal representation of HX in HMAC/SIC encryption mode is:

$c_0 = N \,\|\, m_0 \oplus H((K \oplus opad) \,\|\, H((K \oplus ipad) \,\|\, N \,\|\, i))$

$c_i = m_i \oplus H((K \oplus opad) \,\|\, H((K \oplus ipad) \,\|\, N \,\|\, i))$, to encrypt, and

$m_0 = \Phi_0 \oplus H((K \oplus opad) \parallel H((K \oplus ipad) \parallel N \parallel i))$

$m_i = c_i \oplus H((K \oplus opad) \parallel H((K \oplus ipad) \parallel N \parallel i))$, to decrypt

Finally the formal representation of HX ENVELOPE/SIC encryption mode:

$c_0 = N \parallel m_0 \oplus H(K \parallel \pi_K \parallel N \parallel i \parallel K)$

$c_i = m_i \oplus H(K \parallel \pi_K \parallel N \parallel i \parallel K)$, to encrypt, and

$m_0 = \Phi_0 \oplus H(K \parallel \pi_K \parallel N \parallel i \parallel K)$

$m_i = c_i \oplus H(K \parallel \pi_K \parallel N \parallel i \parallel K)$, to decrypt

The following pictures display the modes of operation for HX

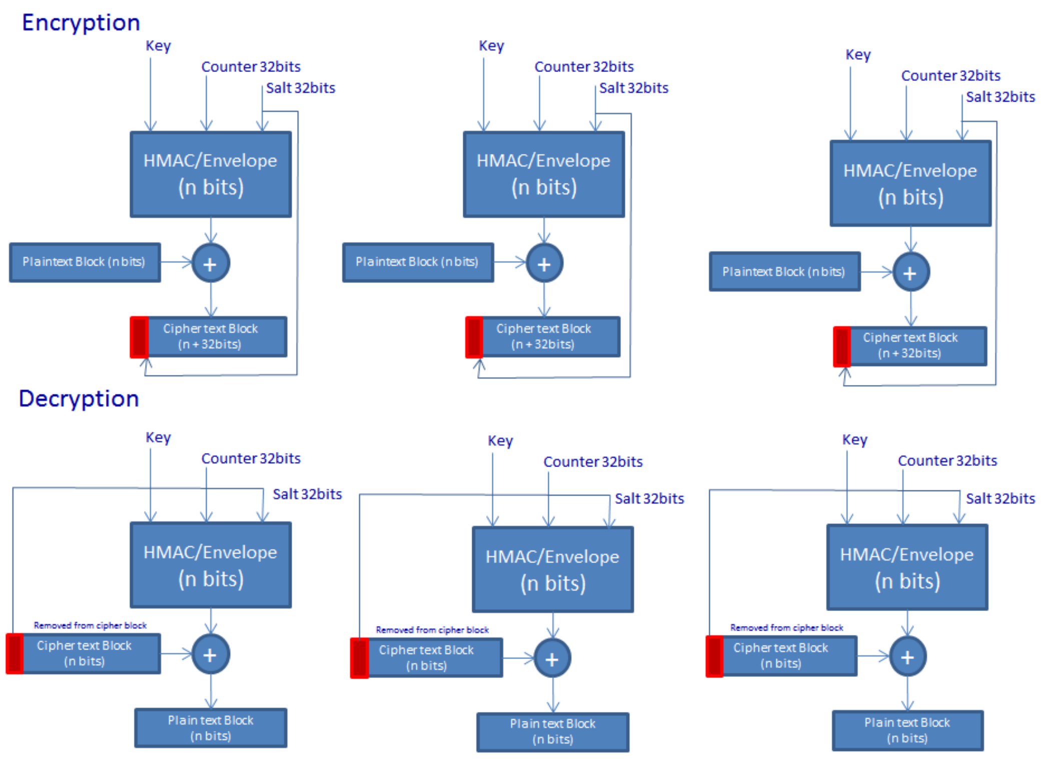


Figure 21 - HX CTR-Mode

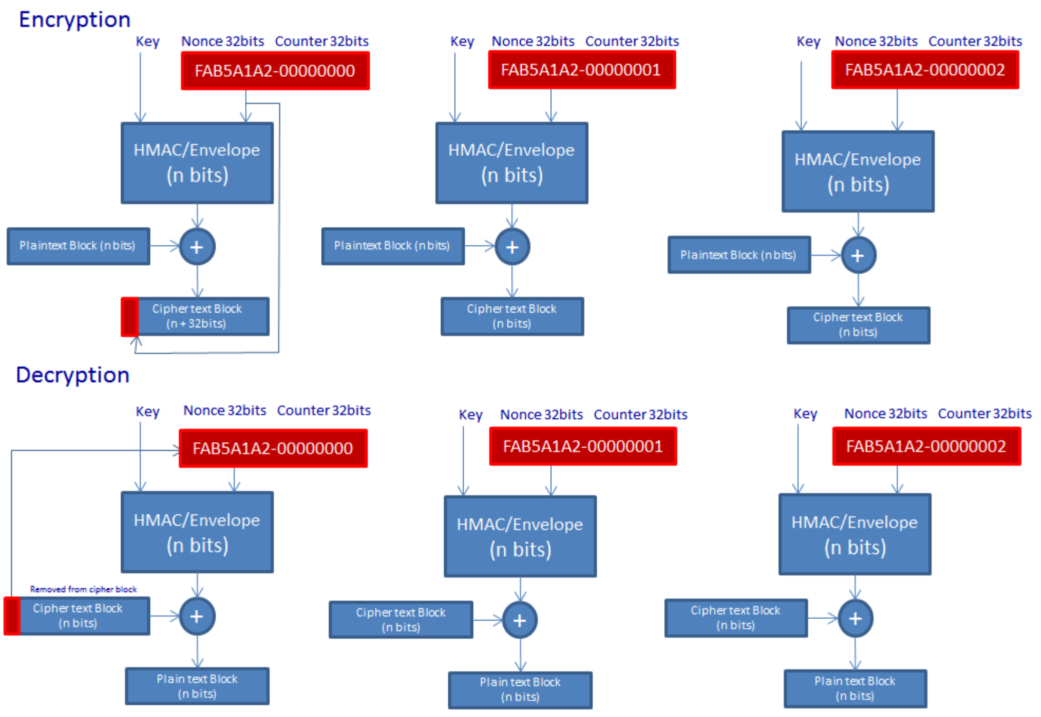

Encryption
Key
Nonce 32bits Counter 32bits
FAB5A1A2-00000000
FAB5A1A2-00000001
FAB5A1A2-00000002
HMAC/Envelope (n bits)
Plaintext Block (n bits)
Cipher text Block (n + 32bits)
Cipher text Block (n bits)
Decryption
Removed from cipher block
Plain text Block (n bits)

Figure 22 - HX SIC-Mode

The table below summarizes the cipher modes for HX and their respective security level. We are considering ENVELOPE less secure than HMAC, for the reason of the best attack against HMAC-MD5 and ENVELOPE-MD5, 2^{97} and 2^{96} respectively [2.6.2] [2.6.3].

Cipher Mode	**Security Level**
HMAC/CTR	Highest
ENVELOPE/CTR	Slightly Less secure than HMAC/CTR
HMAC/SIC	Less secure than the previous ones if the developer uses a constant key
ENVELOPE/SIC	Slightly Less secure than HMAC/SIC

Table 7 - HX Encryption Modes

4.4 Image Encryption Test

We tested the HX encryption algorithm with images. The encrypted image gives an idea of pseudo randomness and the salt cost of the HMAC/CTR.

We converted each pixel into a three Byte string (24 bit integer) containing the RGB values of the pixel. The final plaintext is the concatenation of all three Byte pixel strings. The length of the plaintext is the Width of the image times the Height of the image times 3 Bytes per pixel. After the encryption, a higher image is necessary to store the Salts of the encrypted blocks. Each pixel of the encrypted image takes 3 Bytes of the encrypted message. If, for the final pixel we become shorter than 3 Bytes, the green or blue components of the last pixel are set to zero. As a result, if we only have one Byte for the last pixel, it would have only the red component of the pixel and, if we have two bytes for the last pixel, it would have only the red and the green components. The keystream is generated and a bitwise XOR is applied with the plaintext. Each salt is concatenated with the respective ciphered block to form the ciphered text.

The decryption process is similar. Every pixel has to be converted into a string of RGB components and then the decryption algorithm separates de salts from the encrypted blocks. It then generates the keystream and applies the bitwise XOR operation with the block keystream and the encrypted block to get the plain block. Finally the original image high is calculated from the plain message length and the original image is restored. For image encryption, the salt range is 0 to 2^{24}-1 in order to generate a valid pixel color.

The images below shows an example of an image encrypted with the HX algorithm.

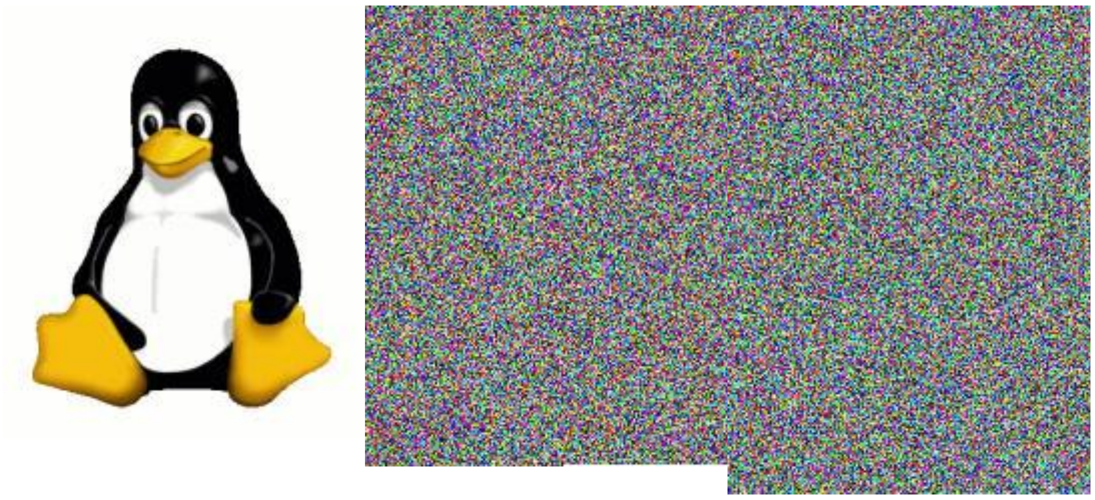


Figure 23- Encryption an image with HX HMAC/CTR

In [

Figure 23], we have the original image (left), the encrypted image using SHA-512 as the keystream generator (center) and the encrypted image using SHA-256 as the keystream generator (right). We can observe the high increase, because of the block salts.

The image below is a test with a 24bit color gradient

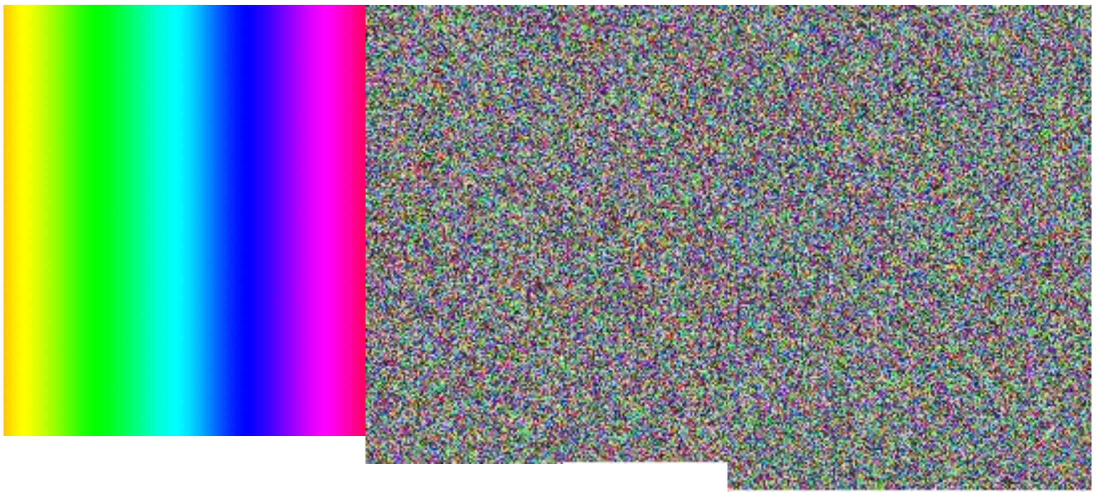


Figure 24 - Encrypting a 24 bit Gradient

The original image (left), the encrypted image using SHA-512 as the keystream generator (center) and the encrypted image using SHA-256 as the keystream generator (right). We can observe that having more colors and less black or white does not affect the pseudo-randomness of the encrypted images.

The image below is a test with a white image

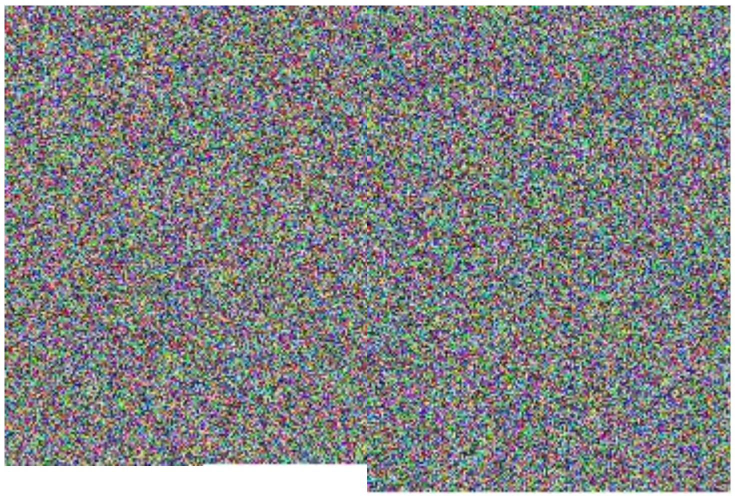


Figure 25 - Encryption of a White Rectangle

The blank original image (left), the encrypted image using SHA-512 as the keystream generator (center) and the encrypted image using SHA-256 as the keystream generator (right). We can observe that having a single color for the entire image does not affect the pseudo-randomness of the encrypted images either.

4.5 HX Authentication Protocol

The HX authentication protocol (HXAuth) uses a long term pre-shared symmetric key or, as an alternative, a password stored on a server repository to authenticate both the client and the server. It is similar to the protocol presented in section [2.10.1], but it takes only two transmissions, while the original protocol takes four instead.

The authentication results in a session key calculated by both endpoints. Such session key will be the key used with HX to provide confidentiality during the session.

Let
r_C: Random from Client in a 32 bit integer positive value converted to string
r_S: Random from Server in a 32 bit integer negative value converted to string
T: Timestamp in a previously agreed format (e.g.: YYYY-MM-DD hh:mm:ss)

H: The CRHF (RIPEMD-160, SHA-256, SHA-384, SHA-512 or Whirlpool)[II]

Id: Client’s URI

LK: Long Term Key negotiated by the 1st time SRAP Authentication or Client’s password stored in the server’s repository

M: The MAC Algorithm. Either HMAC or ENVELOPE

$MAC_H(LK, M_{auth})$: MAC Algorithm digest output, using H, of the long term key and the authentication message

KS: Session Key

```
1   C:   random($r_c$);
         $Msg_1 \leftarrow MAC_H(LK, (Id \| r_c \| T))$;
         $\rightarrow$S: (M, H, $r_c$, T, Id, $Msg_1$);

2   S:   If not validate(M, H, $r_c$, T, Id) then reject C and Stop;
         Else
             $Msg'_1 \leftarrow MAC_H(LK, (Id \| r_c \| T))$;
             If $Msg_1 \neq Msg'_1$ then reject C and Stop;
             Else
                 random($r_s$);
                 $Msg_2 \leftarrow MAC_H(LK, (Id \| r_s \| T))$;
                 $KS \leftarrow MAC_H(LK, (Id \| r_c \| r_s \| T))$;
                 $\rightarrow$C: ($r_s$, $Msg_2$);

3   C:   $Msg'_2 \leftarrow MAC_H(LK, (Id \| rs \| T))$;
         If $Msg_2 \neq Msg'_2$ then reject S and Stop;
         Else
             $KS \leftarrow MAC_H(LK, (Id \| r_c \| r_s \| T))$;
             (Both Client and Server are authenticated)
```

The validate function verifies if all parameters were received, if they are in a canonical format and if they are all valid. It is up to the server to accept or reject the client timestamp based on the time gap between the endpoints’ clocks.

Once both endpoints have been authenticated r_c and r_s will be the nonces for the client and the server respectively. If HX is set to SIC mode of operation. Up to approximately two billion messages can be sent by each side. When either of the counters reaches its last value, the corresponding endpoint sends a REAUTHENTICATION REQUIRED message to the other endpoint. By performing a new authentication, the endpoints will generate another session key and reset both counters.

Since the session key is generated automatically and independently from the will of the developer, it is not possible to use a constant encryption key. This is why SIC mode is a better choice than CTR mode to be used by HXAuth.

[II] Other CRHF such as SHA3-512 or SKEIN-1024 may be added in the future.

Nevertheless it is not recommended to exchange such a huge amount of data with a single encryption key. The endpoints should also negotiate the elapsed session time and Bytes transferred thresholds to trigger a REAUTHENTICATION REQUIRED message to renegotiate a new session key.

4.6 Analysis of HX Authentication Protocol

The strength of the cryptosystem relies on the following assumptions:

1) The CRHF is safe to use, meaning it is not feasible to find preimages, second preimages or collisions.
2) HMAC and ENVELOPE have been proved to be safe [2.6.2] [2.6.3].
3) The timestamp also works as a nonce, making it very difficult for a Replay Attack to succeed [4.6.2.3].

Although HXAuth was designed to work with SRAP, it will also be analyzed for possible vulnerabilities, if it is used outside SRAP, when client’s and server share a common secret such as a pre-shared key or the server has a table of users and their respective passwords stored.

4.6.1 Perfect Forward Secrecy

Perfect Forward Secrecy or simply Forward Secrecy means that a compromised session key should only affect the compromised session, not allowing earlier sessions to be compromised (Gunther, 1990). For example, if an eavesdropper is recording every encrypted session and she was able to guess a specific session key, she can only decrypt the messages exchanged during such specific session. She cannot, however, decrypt the previous sessions.

HXAuth has this property. Each session key generated during the authentication is unique.

4.6.2 Resilience to Protocol Attacks

According to Boyd and Mathuria (Boyd & Mathuria, 2003), an authentication protocol must be resilient against the following type of attacks:

4.6.2.1 Eavesdropping

Unless the selected CRHF is broken, weakened or has a trapdoor function, it is infeasible to find preimages or second preimages of the keys. HMAC and ENVELOPE were proved to be secure, even if they are used with broken CRHF, such as MD5 or SHA1. For an eavesdropper to succeed in braking HXAuth, she has to be able to break HMAC or ENVELOPE with the selected CRHF. Otherwise she has no means to reproduce de session key or deduce the long term key. The probability of guessing the session key is $\frac{1}{\min(2^n, 2^{|K|})}$, where n is the CHRF digest size in bits and |K| is the length of the long term key or pre-shared secret in bits. When using a password as the pre-shared secret, password strength rules apply. For example: a ten Byte password using printable characters only has a strength of 6.57 bits per character, requiring $2^{65.7}$ brute-force attempts.

4.6.2.2 Modification

Modification has the best chance to successfully hijack a session, if the attacker is able to guess the session key. This is why it is prudent to establish elapsed time and Bytes transferred thresholds to mitigate session hijacking.

4.6.2.3 Replay

Depending on the acceptable time gap between the client and the server, a replay attack can be used to try to guess the client’s long term key. During 5 minutes, it is possible to make 300 brute-force attacks, one per second, assuming the server will not allow the same timestamp twice for a given client. As a result, the server must have other mechanisms to protect itself against a prolonged replay attack, like blocking the user’s account if an authentication fails too many times or force a Fast Negotiation authentication (ROSEMBERG, 2014) pp. 65, when HXAuth is being used with SRAP.

4.6.2.4 Preplay

If used as a password based authentication system, HXAuth is vulnerable to preplay attacks, like PHISHING (ROSEMBERG, 2014) pp. 41-43. If the attacker is able to convince the client to put her credentials on a fake system, the attacker will have the user's ID and password to perpetrate a successful attack.

4.6.2.5 Reflection

We do not see reflection as a feasible way to break HXAuth because there is no challenge involved. The client presents her credentials and the server simply replies with a random value and a proof of possession of the shared secret. The session key must be calculated by both endpoints, which cannot succeed without the knowledge of the shared secret.

4.6.2.6 Denial of Service

It is relatively easy to perpetrate a DoS attack on HXAuth. All the attacker has to do is to delay the communications long enough for the time gap between the client and the server to become unacceptable. If this happens, communications should be terminated.

4.6.2.7 Typing Attacks

There is very little information to be used for a typing attack. Using a previously used salt every time is not enough to break the generated keys, since the timestamp cannot be reused and the attacker has no control over the salt generated by the server. However, it is a good policy, not allow the same salts to be used twice in a roll.

4.6.2.8 Cryptanalysis

The strongest point of HXAuth and HX is that cryptanalysis applies to the CRHF, HMAC and ENVELOPE. As long as the MAC algorithms and the CRHF resists key recovery, preimage and second preimage attacks, an attacker will not be able to predict or generate keystreams without the knowledge of the long term key or shared secret.

4.6.2.9 Certificate Manipulation

Certificates are not used in HXAuth; therefore this attack does not apply. Certificates are relevant in SRAP, particularly in the first time authentication if the authentication partner of last resort if used.

4.6.2.10 Protocol Interaction

The user ID is always used to generate the next encryption key. Hence, if multiple users are interacting with a server, each one with a unique user ID, Protocol interaction does not apply. However, if the same user ID is used in different sessions, e.g.: a single pre-shared key, other layer protocols must be able to correctly identify the sessions, not allowing interaction among them.

4.6.3 Advantages of HXAuth

As the analysis showed, HXAuth is much faster than most symmetric key authentication protocols; it is light and easy to be implemented; it is resilient to attacks and can be used in a variety of scenarios well suited for smart cities. It automatically negotiates a session key from 160 to 512 bits or possibly 1024 bits, if Skein becomes a widely adopted CHRF.

The main disadvantage of HXAuth is the endpoints require time synchronization. If the endpoints clocks are out of sync, the authentication will fail and a session key will not be successfully negotiated. However, smart cities of the near future will require more advanced sensors and devices. As a consequence, not only the internal clock of the endpoints will need greater precision but also they will have multiple sources for time synchronization, besides a NTP server, such as GPS time synchronization or synchronization with mobile phone networks. Multiple time synchronization options and a more precise internal clock mitigate DoS attacks by time sync denial.

4.7 HX Class Design

Although the developer can choose a CRHF and a cipher mode, default values must be assigned to both parameters, making them optional, enforcing Kerckhoff's 6th principle [2.1]. From the CRHF set {RIPEMD-160, SHA-256, SHA-384, SHA-512, Whirlpool}, we chose SHA-512 the default CRHF and HMAC/CTR the default cipher mode. This combination provides the strongest keystream possible [2.5.9], [4.2]. Also, according to the NIST, the proper hash digest size for the near future, when smart cities become a reality, is 512 bits [Figure 1]. SHA-512 was preferred to Whirlpool considering the former is faster than the latter [5.4.1] even though they both produce the same digest size.

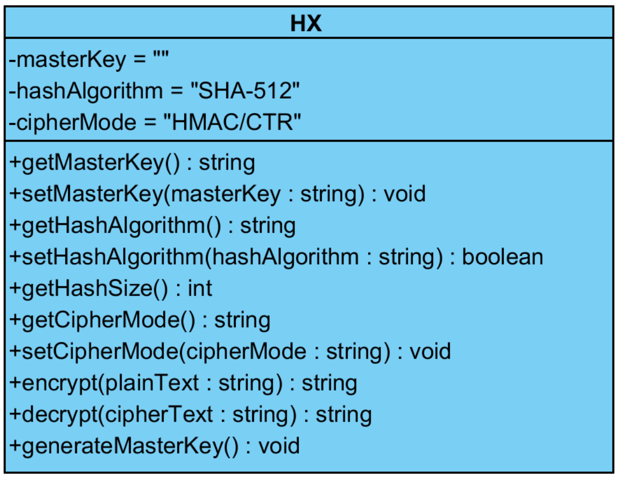

HX
-masterKey = ""
-hashAlgorithm = "SHA-512"
-cipherMode = "HMAC/CTR"
+getMasterKey() : string
+setMasterKey(masterKey : string) : void
+getHashAlgorithm() : string
+setHashAlgorithm(hashAlgorithm : string) : boolean
+getHashSize() : int
+getCipherMode() : string
+setCipherMode(cipherMode : string) : void
+encrypt(plainText : string) : string
+decrypt(cipherText : string) : string
+generateMasterKey() : void

Figure 26 - HX Class Main Attributes and Methods

The attributes and methods presented are the ones developers must be aware of.

A second class, HXBlock was implemented to encrypt and decrypt streams. It is suited to be used by HXAuth protocol. With HX-SIC, every time the encrypt() or decrypt() methods are invoked, the counter is reset and encrypt() generates the nonce. However, for streams, the counter cannot be reset every time a stream is encrypted or the keystream would repeat itself. As a consequence, with HXBlock we added additional methods to handle stream encryption and decryption.

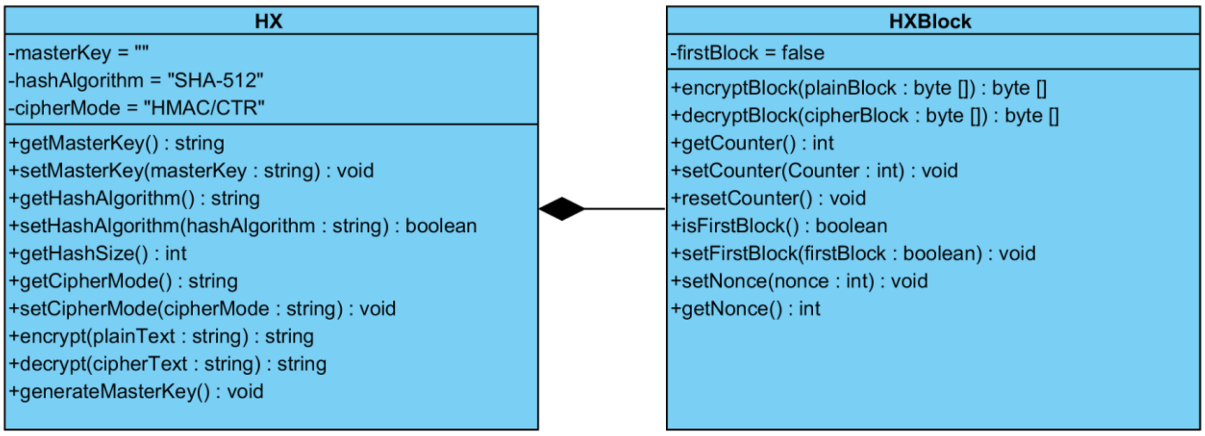

HX
-masterKey = ""
-hashAlgorithm = "SHA-512"
-cipherMode = "HMAC/CTR"
+getMasterKey() : string
+setMasterKey(masterKey : string) : void
+getHashAlgorithm() : string
+setHashAlgorithm(hashAlgorithm : string) : boolean
+getHashSize() : int
+getCipherMode() : string
+setCipherMode(cipherMode : string) : void
+encrypt(plainText : string) : string
+decrypt(cipherText : string) : string
+generateMasterKey() : void
HXBlock
-firstBlock = false
+encryptBlock(plainBlock : byte []) : byte []
+decryptBlock(cipherBlock : byte []) : byte []
+getCounter() : int
+setCounter(Counter : int) : void
+resetCounter() : void
+isFirstBlock() : boolean
+setFirstBlock(firstBlock : boolean) : void
+setNonce(nonce : int) : void
+getNonce() : int

Figure 27 - HXBlock Class Methods

The setFirstBlock() method modifies the behavior of encryptBlock() and decryptBlock(). When firstBlock is false, the counter is incremented each time encryptBlock() or decryptBlock() is invoked. However when firstBlock is true, the counter is reset when encryptBlock() or decrpytBlock() is invoked. The encryptBlock() method generates the nonce and prepend it to the first ciphered block. The decryptBlock() method removes the nonce from the first cipher block and invokes setNonce(). Both encryptBlock() and encryptBlock() methods sets first block to false right before they return the ciphered block or plain block respectively. Both encryptBlock() and decryptBlock() use ENVELOPE/SIC as the cipher mode.

5 Experimental Results

In this section, we report the results from the experiments designed to reject the null hypotheses.

5.1 Survey Results

None of the participants was able to answer all the substantive questions correctly. The study points in the direction most developers do not have enough background in cryptography. They are unable to select a safe encryption algorithm and a safe mode of operation. Even those who believed they have sufficient background or have previously used cryptographic algorithms failed to respond several simple questions about symmetric key cryptography.

5.1.1 Sample Size

Nineteen participants took the survey. Two of them failed to answer all the questions and were eliminated. Six of them answered all the questions but declared explicitly they have no knowledge on cryptography concepts and never used any cryptographic algorithm before. Four had a bachelor’s degree and two had a master’s degree. Nonetheless all 6 were eliminated from the sample. As a result, we ended with an 11 participant sample size.

5.1.2 Sample Qualification

From the 11 participants, 7 had a bachelor’s degree and 4 had a master’s degree. One of the participants had 10 years of experience but never used a cryptographic algorithm in a project before. The participant was not rejected because the participant believed she was fairly familiarized with cryptographic algorithms and was confident she would be able to use them correctly in a project, if she had access to the algorithms’ documentation.

Years of Experience as a Software Developer	Qty.
1	1
3	1
4	2
5	4
10	1

Table 8 – Participants Years of Experience as a Software Developer

Previous experience with cryptographic algorithms (# of Projects)	Qty.
0	1
1	5
2	2
4	1
6	1
7	1

Table 9 - Participants past experience with cryptographic algorithms

How well familiarized are you with symmetric cryptographic algorithms?	Qty.
Not familiarized at all	3
Not well familiarized	3
I’m fairly familiarized	5
Well familiarized	0
Totally familiarized	0

Table 10 - Familiarization Level of the Participants with Cryptographic Algorithms

Tables 8, 9 and 10 qualify the participants. Although 3 of them said they were not familiarized at all with cryptographic algorithms, they have indeed used cryptographic algorithms in at least one project and, as experienced developers, should be able to recognize the best answer of the questions designed to test their knowledge about symmetric cryptography.

5.1.3 Substantive Questions Answers

For each question we highlighted the best possible answer in green and the worst possible answer, when applied, in red.

Which cryptographic algorithm would you choose to encrypt a text file, containing confidential information?	Qty.
DES	0
3DES	0
AES ECB 256	0
AES CBC 128	0
RC4	1
Blowfish	0
I don 't know how to choose	10

Table 11 – Answers from question 5

None of the developers chose AES/CBC with a 128bit key. One chose RC4 because she had used it before. RC4 would have been an excellent choice, if it had not been broken recently [2.4.4.8]. Unfortunately the vast majority answered they did not know how to choose a safe encryption algorithm. This question alone helps to refute H_{02} - The developer chooses AES or Twofish with a mode other than ECB. Nonetheless, additional confirmation will be available from the code produced by the different sample of the controlled experiment empirical study [5.3.2].

When you read SERPENT 256, for you, what is the meaning of the number 256?	Qty.
Key size in Bytes	3
Key size in bits	4
Algorithm's block size in bits	2
Both key size and Algorithm's block size in bits	2
Algorithm's block size in Bytes	0

Table 12 – Answers from question 6

The sixth question aims to check whether the participants understand the difference between the key length and the block size and that the key length is expressed in bits. Thus, they need to know the number 256 expresses the key length in the algorithm SERPENT-256. According to the answers, only 4 out of 11 participants answered the question, correctly. We noticed that many developers, in fact, make some confusion about the key size and the block size. 3 of the participants believed the encryption key size was in Bytes instead of bits.

An incorrect key size in a block cipher algorithm is bound to raise an exception or to return a null string as the resulting ciphertext, depending on the API.

The CBC (Cipher Block Chaining) encryption mode may use and IV (initialization vector). For you, the IV is required or optional and what is its purpose?	**Qty.**
Optional. Strengthening of the cryptographic algorithm	3
Required. Fill the message with the necessary bits to complete the block size	2
Optional. Mislead the Cryptanalyst by putting random information which will not be used in the encryption	1
Optional. If not supplied by the developer, the algorithm will automatically create a random IV	4
Optional. If not supplied by the developer, the algorithm will use a default value for the IV	1

Table 13 - Answers from question 7

Question 7 was tricky. The IV is random information required for the CBC mode of operation, not optional. However its purpose is to strengthen the algorithm by making sure the same plaintext produces a different ciphered text encrypted with the same key. This makes it more difficult for the cryptanalyst to infer relationships between segments of the encrypted message [2.2]. From the answers, we can see there is some confusion between IV and Padding. Almost half of developers think or hope the IV will be managed automatically by the encryption algorithm. Because we had two questions to be considered with only one choice, we decided to ask the developers for the best answer.

In your understanding, what is the purpose of Padding in block cipher algorithms?	**Qty.**
Strengthening of the cryptographic algorithm	2
Fill the message with the necessary bits to complete the block size	7
Mislead the Cryptanalyst by putting a random information which will not be used in the encryption	1
Optional. If not supplied by the developer, the algorithm will automatically create a random Padding	1
Optional. If not supplied by the developer, the algorithm will use a default value for the Padding	0

Table 14 - Answers for Question 8

This question had the objective to test if developers understand the concept of padding. The majority answered correctly. Nevertheless to pad means exactly to fill or to cover something. The developers might have deduced correctly the purpose of padding by the answers.

In relation to cryptographic keys and block cipher algorithms, what is your understanding about the key size?	Qty.
The grater the key size, the better	2
The key size must be exactly that same size of the block of the algorithm	2
The key size must be exactly the size in bits specified by the algorithm, not matter the block size	3
If the key size is greater than the algorithm’s block size, the algorithm will truncate the key to the block size automatically	3
If the key size is less than the algorithm’s block size, the algorithm will automatically concatenate the key with the necessary blanks (spaces) needed to match the block size	2

Table 15 - Answers for Question 9

The ninth question explores even further the knowledge of the participant about the key sizes in block cipher algorithms. Only 3 out of 11 participants chose the correct answer. 4 participants think encryption algorithms automatically handle keys that do not comply with the specifications. Not only that is not true but it is also a dangerous assumption. If an algorithm requires a 128 bits long key (16 Bytes) and the developer passes a key of only 64 bits (8 Bytes) long, assuming the encryption algorithm would “padd” the last 64 bits of the encryption key, that action would substantially decrease the security of the encryption. 2 of the participants believe the greater the key size, the better, which is the principle of the Vernam Cipher [2.3]. However, such concept does not apply to block cipher algorithms. Finally, 7 of the participants confused the block length with key length, which are two distinct concepts.

Concerning cryptographic Keys and IVs, what must be secret and what must be transmitted openly with the ciphertext?	Qty.
Both must be secret. The Key and the IV must be negotiated in a secure channel	4
Both must be transmitted openly with the ciphertext	0
The Key is secret and must be negotiated in a secure channel. The IV must be transmitted openly with the ciphertext	4
The IV is secret and must be negotiated in a secure channel. The Key must be transmitted openly with the ciphertext	2
Both must be secret. The HASH of the IV with the key must be informed	1

Table 16 - Answers for Question 10

The tenth question checks whether the participant knows exactly what information should be transmitted in the open and what should be negotiated in a secure channel: the encryption key or the IV. According to the answers, 4 out of

11 participants answered correctly. The key is secret and must be negotiated in a secure channel. However, the IV is not and is transmitted openly with the ciphertext. 4 answered both the IV and the Key must be secret. Albeit this policy does not compromise security, it defeats the purpose of the IV [2.2]. 2 of the participants even said the key must not be secret. Only the IV should be. The participants who chose this answer, revealed a complete lack of understanding of the most basic cryptography concept.

Concerning Padding Schemes, observe the following explanation about Padding and choose the safest scheme in your opinion	**Qty.**
I prefer ANSI X.923 because of its simplicity and confusion avoidance between valid nulls (nulls which are part of the original message) and invalid nulls (the ones that are padded to the message)	2
I prefer ISO 10126 because random bits will make it more difficult for a cryptanalyst	4
I prefer PKCS7 because it is even more guaranteed to identify valid and invalid bits (padded bits)	1
I prefer filling the message with Nulls or Blanks because there is nothing simpler than these schemes	0
I prefer a scheme that exists on any API in order to guarantee interoperability	4

Table 17 - Answers for Question 11

About question 11, there is no correct answer. Padding is used on the last block of the message. If the message length is a multiple of the block size, an entire block of Padding is added to the message. The most significant answers (*I prefer a scheme that exists on any API in order to guarantee interoperability* and *I prefer ISO 10126 because random bits will make it more difficult for a cryptanalyst*) are exactly what we hoped to get. ISO 10126 is the most secure, because it uses random Bytes for the Padding. The other padding schemes may be vulnerable to the Padding Oracle Attack (Manger, 2001 pp. 230-238), if the "oracle" (usually a server) leaks data about whether the padding of an encrypted message is correct or not. Such data can allow attackers to decrypt or encrypt messages through the oracle using the oracle's key, without the knowledge the encryption key.

PKCS7, ISO 10126 and ANSI X.923 are interoperable. The last Byte on the 3 schemes determines the length of the Padding. Once decrypted it will tell the algorithm how many Bytes of the decrypted message need to be discarded. Nevertheless, if blanks or nulls are used, a binary message my not be decrypted correctly. Besides, when blanks or nulls are used, no padding occurs if the message length is a multiple of the block size. None of the participants chose Blanks or Nulls as their Padding schemes. Their comments confirm their primary concern was interoperability and security.

From to the answers we collected, the only concept of block cipher encryption developers from our sample seem to understand is Padding. However, confusion between key sizes and block sizes and the lack of understanding of the purpose of the IV, points in the direction that most developers are unable to use a block cipher algorithm, like AES, correctly.

Considering none of the developers was able to answer all of the substantive questions accurately, even though 10 out of 11 have had previous experience with cryptographic algorithms and 5 out of 10 declared they were fairly familiarized with cryptographic algorithms, we can also conclude they lack the necessary background in symmetric key cryptography, thus rejecting H_{01}.

5.2 Keystream Generation Experiment

We selected five CRHF for testing purposes: RIPEMD160, SHA-256, SHA-384, SHA-512 and Whirlpool. For each CRHF, we generated one billion keys, divided in 200 sets of 5 million keys. We used a HMAC [2.6.2] and ENVELOPE [2.6.3] algorithms as pseudorandom algorithms to generate the keystreams to emulate the Vernam Cipher [2.3], with each salt ranging from 0 to 2^{31}. The counter started at 0 and was incremented by one for each keystream generated. The first set had counters ranging from 0 to 4,999,999. The second had counters ranging from 5,000,000 to 9,999,999, and so on.

The generated keystreams were validated for uniqueness in a key-value structure. An attempt to insert a duplicated keystream into the set would raise an exception and would have not rejected the null hypothesis.

Because of memory constraints, each set was saved in a CSV (comma separated values) file and loaded into memory only when necessary. Each CSV file contained the keystream in a hexadecimal format, the salt and the counter, both 32 bit integers converted to strings. Two CSVs at a time were loaded into a key-value structure. Again, a duplicate keystream would cause an exception and the null hypothesis would not be rejected. All possible file combinations were tested.

For all CHRF, we tested the basic MAC, HMAC and ENVELOPE as keystreams generators.

We used the string: "Key_&_TesT-2017" without the quotes for the secret key. The cost of a brute force attack on this key is $2^{98.55,}$ since each character belongs to the printable character set (95 characters = $2^{6.57}$) and the length of the key is 15 characters.

After the generation of the sets, each set was loaded into memory and confronted with the others for duplicated keys, two at a time. As expected, there were no key duplications and the null hypothesis (H_{04}) was rejected for all CRHF and MAC algorithms combination.

One billion keystreams is a tiny fraction of the $2^{n/2}$ number, which keystreams duplication is expected. However in a real life application, a session key is not expected to last long enough to transmit or receive 32GB of data, using SHA-256 to generate keystreams, or 64GB of data, when using SHA-512 instead. As a result, 1 billion unique keystreams is large enough to refute $H_{04.}$

The entire collection of sets requires more than 1.5TB. They have been preserved for data provenance and will be made available for download upon request.

The basic MAC algorithm was tested to prove any CRHF can be used to generate keystreams, using the key as the seed. However, none of the CRHF tested should use the basic MAC as a keystream generator in a real life application, since they all use the Merkle-Damgård construction, which is vulnerable to the Length Extension Attack [2.5.1].

5.3 Controlled Experiment Empirical Study

Twelve participants took part in the experiment. Half of them were asked to start with HX and the other Half with an algorithm of her choice. **The sample of this experiment is not the same of the survey experiment.**

One of the participants (number 5), which started with AES-CTR, failed to implement a code using HX to encrypt and decrypt messages. The participant modified the code from the previous tasks and, although HX was instantiated and the masterKey was set, the encryption and decryptions methods were called from the previous experiment. Not from the HX class.

One of the participants (number 8) which started with HX, was not able to complete the task with the algorithm of her choice. Although the participant is a veteran developer, she is a bearer of special needs. Having a 100% visual impairment, the developer had to use an application to read the words of the IDE and console and speak them into her earpiece. Also, she had to hear every single word from the encrypted message, which was time consuming and irritating. She gave up after one hour. She chose AES-CBC as a result from a search engine query. However, she produced the simplest code from all participants, when HX was used, without violating any cryptographic rules.

5.3.1 Sample Qualification

From the twelve participants, in regard to the highest academic degree, half had a master’s degree, five had a bachelor’s degree and one had a high school degree.

In regard to the experience as a software developer, we had a range in years varying from 4 to 40 with an average of 12.6 years and 9.5 years as the median. The table below displays the experience in software development of the participants.

Participant	Years of experience as a software developer
8	40
7	34
12	4
3	9
4	11
11	10
9	7
1	7
5	5
2	4
6	10
10	10
Mean	**12.6**
Med	**9.5**

Table 18 - Sample Distribution of the Years of Experience as a Software Developer

Half of the participants had never used cryptographic algorithms. From the other half, one participant had used cryptographic algorithms in one project, four had used cryptographic algorithms in two projects and one had used cryptographic algorithms in five projects.

Number of Projects using Cryptographic Algorithms	Participants Qty.
0	6
1	1
2	4
5	1

Table 19 - Previous Experience with Cryptographic Algorithms

All of the participants have, at least, some familiarity with Java. None failed to complete the tasks due to a lack of familiarity with Java.

Only two of the developers declared they had previous training in cryptography. However both were self-taught in cryptography. None have had any formal training in cryptography.

When we asked the developers about who, in their organization, was responsible to inspect the code in order to detect security vulnerabilities, half of the participants answered nobody had such a responsibility.

Professional Responsible for Inspecting the Code for Security Vulnerabilities	Qty.
None	6
The developer	1
The project leader or manager	3
A security expert (either internal or outsourced)	2

Table 20 - Professional responsible for vulnerabilities in the code in the organizations of the participants

However, when we asked the participants about who should be responsible to check or detect violations in encryption rules, only two of the participants believe the developer should be the one responsible. The others believe an expert, a plug-in or the class itself, chosen to encrypt and decrypt messages, should have that responsibility.

Who Should Be Responsible for Encryption Rules Violations Checks	Qty.
The developer	2
A plug-in or external tool	1
The project leader or manager	1
A security expert (either internal or outsourced)	4
The Class chosen for encryption or decryption	4

Table 21 - Professional who should be responsible for encryption rules violations

5.3.2 Experiment Results

After analyzing the code produced by the participants and verifying their answers of the feedback form, we were able compare the effectiveness and the efficiency of HX in relation to the both the chosen algorithm and AES-CTR algorithms. We used the following criteria:

1) Safe Encryption Algorithm: Since none of the participants chose Twofish, any participant who did not choose AES as her encryption algorithm, were automatically marked as unable to choose a safe encryption algorithm. Nevertheless, if a participant chose AES, we did not automatically mark

her as someone able to choose a safe encryption algorithm. Depending on how she answered the feedback form question of why she decided to choose the chosen algorithm, she may have been marked as unable to choose a safe encryption algorithm. We had answers saying AES was chosen because it was the first the developer found in a search engine query about encryption. From that particular answer we inferred the participant was unable to choose a safe encryption algorithm. Still, participants who said they have used AES before or they researched and found an article saying AES was safe or the most used were marked as able to choose a safe encryption algorithm. We defined this criterion as the conscious choice in the table results.

2) If the developer chose a trivial key, the plaintext as the key, her own name as the key, a key which is likely to be in a dictionary or a key with a complexity less than 2^{64} bits, such choice was marked as a weak key violation. We had one special case where the developer generated a random encryption key but used it as the IV. Since IVs are transmitted in the open, the participant was in fact disclosing the encryption key. Since disclosing the secret key is the worst possible mistake, we decided to mark such mistake as a weak key violation with a key complexity of 2^0.

3) If the developer chose EBC mode of operation, either because she chose AES with default options or explicitly declared ECB mode, such decision was marked as an unsafe operation mode violation.

4) If the developer used a constant IV in CBC or CTR modes of operation, this was marked as a constant IV violation.

5) If the developer chose AES/CBC for the free choice algorithm task but did not make any modification in the code other than change the instance from (AES/CBC/PKCS5Padding) to (AES/CTR/NoPadding) in the AES-CTR task, the time taken to implement the free choice task was used as the time of AES-CTR task. Otherwise, the real time spent in the AES-CTR task was used.

The participants numbered 1 to 6 started the experiment with the free choice of the algorithm task. The participants numbered 7 to 12 started the experiment with the HX task.

5.3.2.1 Effectiveness Test Results

Based on the criteria previously mentioned and the analysis of the code produced by the participants, we compiled the following tables:

				Cryptographic Rules Violations					
Participant	**Choosen Algorithm**	**Weak Alg**	**Conscious Choice**	**Weak Key**	**ECB Mode**	**Constant IV**	**Total Violations**	**Notes**	**Key Complexity**
1	AES/ECB	0	0	1	1	0	2	Dictionary	2^83
2	AES/CBC	0	1	0	0	1	2		2^75
3	AES	0	0	1	0	1	2		2^41,5
4	AES/CBC	0	1	0	0	1	2		2^128
5	AES/CTR	0	0	1	0	1	2	Used the Key as IV	2^0
6	AES/CBC	0	1	0	0	1	2		2^106
7	DES	1	0	1	1	0	3		2^41,5
9	AES/CBC	0	0	0	0	0	0		2^128
10	AES/ECB Default	0	1	0	1	0	2		2^101
11	AES/ECB Default	0	0	1	1	0	2	Own Name as Key	2^86
12	AES/ECB Default	0	0	1	1	0	2	Trivial Key	2^75
8	AES/CBC	0	0					Failed to Complete	
	Totals	1	4	6	5	5	21		

Table 22 - Free Choice Algorithm Test Results

From the table above, we can see that all but one of the participants violated, at least, one rule, with an average of 2 violations per participant and 21 violations in total. This number reinforces the conclusion developers do not have the basic knowledge to use cryptographic algorithms and reinforces the rejection of H_{01}.

Four of the participants were able to choose a safe encryption algorithm but eight weren’t. Of those who did choose a safe encryption algorithm, two declared they have had previous training in cryptography. With only 1/3 of the participants were able to choose a safe encryption algorithm and none finished the task without a violation (other than a weak key), we can conclude, in general, developers do not know how to consciously choose a safe encryption algorithm and therefore H_{02} is rejected.

Participant	Weak Key Violation	Constant IV Violation	Notes	Key Complexity
1	0	0	Failed to Complete	2^83
2	0	1		2^75
3	1	1		2^41,5
4	0	0	Failed to Complete	2^128
5	1	1	Used the Key as IV	2^0
6	0	1		2^106
7	0	0		2^83
9	1	1	Trivial Key	2^64
10	0	0		2^101
11	1	1	Own Name as Key	2^86
12	0	0	Failed to Complete	2^83
8			Failed to Complete	

Table 23 - AES-CTR Task results

In relation to the AES-CTR task, only two of the participants were able to complete the task without violating the non-random IV rule, which is critical in CTR mode [2.4.3.5]. They also managed to choose an encryption key strong enough not to be found by a dictionary attack or by brute force attack. Four of the participants failed to complete the tasks. However participants 4 and 12 would have succeeded in completing the task if they had instantiated the Cipher class with (AES/CTR/NoPadding) instead of (AES/CBC/PKCS5Padding). The results leads us into the conclusion developers are not able to use AES-CTR, thus rejecting H_{03}.

In relation to HX, the only rule developers can violate is the choice of a weak key, which is exactly what Kerckhoff defined as an ideal cryptosystem [2.1], in terms of usability.

Participant	Weak Key Violation	Notes	Key Complexity
1	1	Trivial Key	2^83
2	0		2^422
3	1		2^41,5
4	0		2^128
5	0	Failed to Complete	2^0
6	0		2^422
7	1		2^36
9	1	Trivial Key	2^64
10	0		2^422
11	1	Own Name as Key	2^38
12	1	Used plainText as Key	2^83
8	0		2^240

Table 24 - HX Task Results

As stated before, the participant number 5 failed to complete the HX encryption and decryption task. All others were able to complete the task. Still, half of the participants violated the **Do not use a weak key rule**. Six participants either generated their encryption keys or chose a strong encryption key. All consoles outputs of the codes execution of the participants who successfully completed the task, show different ciphered texts for the same plaintext encrypted with the same key on each of the five iterations.

The following table compares the total number of cryptographic rules violations on all three tasks.

Task	# of Violations	# of participants who completed the task	Violations/Participant
Free choice algorithm	21	11	1.91
AES-CTR	10	8	1.25
HX	6	11	0.55

Table 25 - Total Cryptographic Rules Violations Comparison

From

Table 25], we can verify developers violate less cryptographic rules with HX than they do with other algorithms, thus we can reject H_{05}.

5.3.2.2 Efficiency Test Results

In accordance with the criteria defined in [5.3.2], we produced the following tables:

Participant	Start Time	End Time	Time Spent	Notes
1	18:44	19:28	00:44	
2	10:00	11:10	01:10	
3	09:30	11:26	01:56	
4	12:40	13:05	00:25	
5	18:05	18:45	00:40	
6	20:28	21:16	00:48	
7	09:45	10:20	00:35	
9	16:15	16:41	00:26	
10	20:00	20:28	00:28	
11	12:22	12:35	00:13	
12	09:47	10:15	00:28	
8				Failed to Complete

Table 26 - Free Choice Algorithm Time Spent on Task

In the Free Choice Algorithm task, the total time spent in the implementation of the code by the 11 developers who successfully completed the task was 7 hours and 53 minutes. The average time of the task was 43 minutes.

Participant	Start Time	End Time	Time Spent	Notes
1				Failed to Complete
2	10:00	11:10	01:10	
3	09:30	11:26	01:56	
4				Failed to Complete
5	18:05	18:45	00:40	
6	20:28	21:16	00:48	
7	10:21	10:52	00:31	
9	16:42	16:59	00:17	
10	20:34	20:59	00:25	
11	12:35	12:52	00:17	
12				Failed to Complete
8				Failed to Complete

Table 27 - AES-CTR Time Spent on Task

In the AES-CTR task, the total time spent in the implementation of the code by the 8 developers who successfully completed the task was 6 hours and 4 minutes. The average time of the task was 45 minutes.

Participant	Start Time	End Time	Time Spent	Notes
1	19:50	20:05	00:15	
2	11:30	11:40	00:10	
3	11:31	11:43	00:12	
4	13:14	13:20	00:06	
5				Failed to Complete
6	21:53	22:10	00:17	
7	09:16	09:43	00:27	
9	15:45	16:12	00:27	
10	19:37	19:44	00:07	
11	12:16	12:22	00:06	
12	09:21	09:47	00:26	
8	09:45	10:25	00:40	

Table 28 - HX Time Spent on Task

In the HX task, the total time spent in the implementation of the code by the 11 developers who successfully completed the task was 3 hours and 13 minutes. The average time of the task was 17 minutes.

Task	# of participants who completed the task	Total Time Spent	Average Time/Participant
Free choice algorithm	11	7:53	43’
AES-CTR	8	6:04	45’
HX	11	3:13	11’

Table 29 - Efficiency Test Comparison

From Table 29, we can verify developers take less time to implement an encryption and decryption application with HX than they do with other encryption algorithms, even without example codes available to copy and paste. HX is more efficient than the other algorithms in the usability context. As a result, we can reject H_{06}.

5.3.2.3 Feedback Form Additional Information

All of the participants declared they copied and pasted code from a forum or from some other documentation in order to complete the free choice algorithm task. The participants 3, 4, 6, and 9 declared they checked the code for some vulnerability. Nevertheless only the participant 9 was able to complete the task without any violation. Even so, she did not specify a random IV. She left it up to the Cipher class default behavior, which is to randomize the IV.

The participant’s behaviors were repeated with the AES-CTR task. The participant 2, however, did not copy any code from forums. All she did was to change the instance of the Cipher class from (AES/CBC/PKCS5Padding) to (AES/CTR/NoPadding). The participant 9, who completed the free choice algorithm task without any violations, chose a weak key and a constant IV in the AES-CTR task. The participants 7 and 10, which were the only ones who completed the AES-CTR task successfully, did copy and paste code but did not checked the code for vulnerabilities.

None of the participants opted to change the default hash algorithm and cipher mode of the HX class. Only one participant (number 10) declared she examined the options and decided the default values were the “best” choices.

About the question that asked if the participant found strange the fact the encrypted message from the AES-CTR task was different each time the plaintext was encrypted, even though the key was the same, only two participants answered it was supposed to be this way. This is due the fact the CTR mode of operation requires a nonce [2.4.3.5]. Nonetheless the participant 7 reported she chose AES because it was the first algorithm found in a Google query. She did not make a conscious choice on the encryption algorithm. In addition, she only changed the instance of the Cipher class form CBC to CTR to complete the AES-CTR task. We deduce she does not know the purpose of the IV but was lucky to find a good example from a forum. Nonetheless, the participant 10 did make a conscious choice in the AES algorithm and produced different codes for the free choice and AES-CTR tasks. She noticed the HX produced different ciphered texts for the same plaintext and key on each loop and associated it as the right behavior for an encryption algorithm. If she started the test with the free choice algorithm first, the answer could have been different.

We did not expect half the participants would choose weak keys for the HX experiment. We hoped all the participants would have thought of the key like a password and choose a strong password or passphrase for the key or that they would choose to invoke the generateMasterKey() method.

5.3.2.4 Qualitative Analysis

From the experiment results, the qualification and the feedback forms, we compiled the following table:

Participant	Analysis
P1	The participant copied and pasted the code used in the experiments, claiming it was the first example she found in a web search. She did not check the code for vulnerabilities. She said "AES-CTR is not safe, because it is too difficult to use". She said she did notice the encrypted message was different for each loop in AES-CBC, but it was not. She confused the console output from the HX task. She had no idea why the ciphered texts were different in the HX experiment. From that statement, we can also infer she does not know the purpose of the IV. Even having copying and pasting code from forums, she found the experiment difficult to complete. She violated rules the do not use ECB mode and the do not use a weak key. She has a Master's degree, seven years of experience in software development and used encryption algorithms in five projects. She claimed she was highly familiarized with the Java language and had no formal training in cryptography. With HX, she violated the rule do not use a weak key. Conclusion: we are talking about a specialist in computer science, with experience in software development, previous experience with cryptographic algorithms and good knowledge on the Java language. However she does not seem to understand the basics of cryptography.
P2	The participant copied and pasted the code used in the experiments. She

	claimed she had previous experience with AES, nevertheless she did not check the code for vulnerabilities. She used a constant IV on both free choice and AES-CTR task. Even with four years of experience in software development, having used cryptographic algorithms two projects, she claimed the tasks were relatively hard to complete. She had a bachelor's degree in informatics and when asked if the ciphered texts were different on each loop, she answered in her code it did not happen. She made conscious choices on the algorithm and mode of operation, did not choose a weak key, had not formal training in cryptography and was highly familiarized with the Java language. Conclusion: the subject demonstrates concerns only about the strength of the key. She seems to have chosen AES-CBC by chance and does not seem to know the purpose of the IV. Besides the need of a strong encryption key, the participant did not show good knowledge of best practices in cryptography.
P3	The participant copied and pasted the code used in the experiments, claiming it was the first example she found in a web search. She did not check the code for vulnerabilities. When asked if the ciphered texts were different on each loop, she answered in her code it did not happen. Even having copying and pasting code from forums, she found the experiment difficult to complete the first task but not the second. She violated no rules on the first task but chose a weak key and a constant IV for the AES-CTR task. She has a bachelor's degree, nine years of experience in software development, not having used encryption algorithms in any project before. She claimed she was totally familiarized with the Java language but had no formal training in cryptography. With HX, she violated the rule do not use a weak key. Conclusion: this subject is a senior Java developer, with lots of experience in software development, who seem to have accomplished the first task by luck in choosing a secure code from a forum. Nevertheless she does not seem to understand the basics of cryptography.
P4	The participant copied and pasted the code used in the experiments, claiming it was conscious choice because she is well known about cryptography principles. She did check the code for vulnerabilities. Nevertheless she failed to complete the AES-CTR task and violated the do not use a constant IV rule. She said it answered it does not matter if the ciphered texts are always the same. What matters is the message was correctly encrypted and decrypted. From that statement, we can also infer she does not know the purpose of the IV. Even having copying and

	pasting code from forums, she did not find the experiment difficult to complete. She has a Master's degree, eleven years of experience in software development and used encryption algorithms in one project. She claimed she was totally familiarized with the Java language and had formal training in cryptography as a self-taught learned in the domain. With HX, she violated no rules Conclusion: we are talking about a specialist in computer science, with lots of experience in software development. It was a surprise the experiments showed she does not know the basic concepts of cryptography, except the need of a strong key, even though she taught she did.
P5	The participant copied and pasted the code used in the experiments, claiming it was the first example she found in a google search. She did not check the code for vulnerabilities. When asked if the ciphered texts were different on each loop, she answered in her code it did not happen. She found the experiment relatively easy to complete, although she took the longest time to complete the experiment and made the worst possible mistake: used the key as the IV, even though the key itself was not weak. She also failed to complete the HX task, because she instantiated the HX class, set the masterKey but called encryption and decryption methods from the previous task, probably due to fatigue. We have no doubt she would be able to complete the HX task without any violations if HX was the first task. She has a bachelor's degree, five years of experience in software development, not having used encryption algorithms in any project before. She claimed she was fairly familiarized with the Java language but had no formal training in cryptography. Conclusion: at the time of this work, this subject was taking the final semester to get her Master's degree. She has a significant experience in software development. However she demonstrated to be unable to work with cryptography.
P6	The participant copied and pasted code from forums, but claimed she did check the code for vulnerabilities. Having a Master's degree, she did what any researched supposed to do: she quickly researched about encryption algorithms and chose AES-CBC for the first task. She chose a strong key but used a constant IV. On the HX task, she was one of the two who invoked the method generateMasterKey() and produced the strongest possible key, completing the HX task with no violations. Having ten years of experience in software development, without previous experience with cryptographic algorithms, she claimed she was highly familiarized with the

	Java language and had no problems with the implementation of the tasks. Conclusion: although this subject has no formal training in cryptography, she was able to complete the HX task without aby violations and was able to maximize the security of the encryption. However with a more difficult to use API, she was not able to complete the tasks without violations.
P7	The subject copied and pasted code from forums, claiming it was the first code that appeared after a web search. She did not check the code for vulnerabilities. She assumed the code from the forum was safe. She did notice the encryption was different on each encryption loop, but did not know why it happened, demonstrating she does not know the purpose of the IV and completed the AES-CTR task not violating any rule by luck. However, on the second task, she was not lucky and used an unsafe code which used DES mode of operation and a weak key. The same weak key was used in the HX task. The participant had a bachelor's degree in Computer Science and a *Latu-Sensu* specialization course. She has thirty four years of experience in software development and claimed she used the Java language professionally, even though she did not work with encryption algorithms before. Conclusion: experience in software development had no impact on security issues. The participant assumed any code posted in a forum, which was not refuted by another developer is secure. A dangerous assumption.
P8	Even though this subject has visual impairment, she was able to complete the HX task, producing the simplest code. Her disability, however, impacted on the other tasks. Having the need to use an earpiece to read the console output and the hexadecimal ciphered texts caused frustration and fatigue. She gave up after one hour. The participant has a high school degree and forty years of experience in software development, although she has not used encryption algorithms previously. She was totally familiarized with the Java language and had no formal training in cryptography. Conclusion: an easy to use API helped the subject with complete the HX task, while a more complicated API, even with the help of codes from forums was more difficult to handle.
P9	The subject copied and pasted code from forums, claiming it was the first code found on a google query. She said she checked the code for vulnerabilities and the code was successful in completing the first task without any violations. On the AES-CTR task, however, the code copied,

	and checked for vulnerabilities, produced the worst results. The key was weak the IV was fixed. Even though she claimed she noticed the ciphered texts were different after each loop, she declared it does not matter if the ciphered texts are constant or different. What matters is the proper encryption and decryption. The subject has a Master's degree and seven years of experience in software development. She had no previous experience with encryption algorithms, is totally familiarized with the Java language but had not formal training in cryptography. She completed the HX task, violating the not to choose a weak key rule, the same she used in the AES-CTR task. Conclusion: The subject does not seem to take privacy and security seriously or she assumes a security issues are the responsibility of another professional, either internal in the software house or an outsourced security expert.
P10	The subject copied and pasted code from forums, claiming she checked the code for vulnerabilities. Even though she violated the IV rule on the free choice algorithm, she implemented a very robust code to generate pseudo-random nonces for the AES-CTR task and completed that task without any violations. The participant has a Master's degree, ten years of experience in software development and she had used encryption algorithms in two previous projects. She is totally familiarized with the java language and claimed she had formal training in cryptography. She completed the HX task without any violations and she was the other participant who invoked the generateMasterKey() method, producing the strongest ciphered texts. Conclusion: even for an experienced developer with formal training in cryptography, it is difficult not to violate any best practice rule in cryptography. This emphasizes the need of a simple and easy to use encryption API, which automatically sets maximum security settings by default upon instantiation.
P11	The participant copied and pasted the code used in the experiments, claiming it was the first example she found in a web search. She did not check the code for vulnerabilities. When asked if the ciphered texts were different on each loop, she answered she did not notice and it does not matter. What matters is to encrypt and decrypt the message correctly. She found the experiment easy to complete. the first task but not the second. She violated no rules on the first task but chose a weak key and a constant IV for the AES-CTR task. She has a Master's degree, ten years of experience in software development, having used encryption algorithms

	in two projects before. She claimed she was fairly familiarized with the Java language but had no formal training in cryptography. With HX, she violated the rule do not use a weak key. She used her own name as the key on all experiments. With the free choice task, she chose AES-ECB and on the AES-CTR used a fixed IV. Conclusion: The subject is a specialist in computer science, with experience in software development, previous experience with cryptographic algorithms and fair knowledge on the Java language. However she does not seem to understand the basics of cryptography.
P12	The participant copied and pasted code from forums on both task1 and task2. She did not check the code for vulnerabilities and justified the codes were found on a search query. She chose AES-ECB (default instantiation of the Cipher class) with a trivial key. She failed to complete the second task, because she instantiated AES-CBC instead of AES-CTR. Even if she instantiated the class correctly she would have used a fixed IV and the same trivial key. On HX, she also used the same weak key. When asked about the different ciphered texts on each loop, she answered it is supposed to be this way, even though in her code the ciphered texts were the same on every loop. She has a bachelor’s degree, four years of experience in software development and claimed she used cryptographic algorithms in two previous projects. She has no formal training in cryptography but is highly familiarized with the Java language. Conclusion: the participant either does not understand the basic principles of cryptography or she did not take the experiment seriously.

From the empirical studies, we can conclude developers from the sample misuse cryptographic APIs due to a lack of formal training. They confuse basic concepts such as an IV and padding, cannot choose a safe encryption algorithm and same do not even understand the importance of the encryption key. Although the sample sizes of both the survey and the controlled experiment were small, recent studies confirm our conclusions. Braga and Dahab found out cryptography misuse is common in on-line communities and are recurrent in developer’s discussions. The authors concluded developers learn to use cryptographic APIs without learning the tricky details of cryptography (Braga & Dahab, 2017). In their experiments, the authors observed experienced developers using fixed IVs and some developers using obsolete encryption algorithms.

Other researches also conducted experiments to understand security mistakes made by developers. In a recent study, the authors observed 53% of the developers used a fixed IV or a not random enough IV, or the use of a weak encryption algorithm (Votipka, et al., 2019).

5.4 Performance Tests

In addition to the experiments designed to prove the research questions, we designed two other test scenarios, where we compared the performance of HX with AES, first encrypting and decrypting messages in a computer and second in a VPN simulated environment, where we measured how the encryption affected network performance at different network speeds.

Initially we made a comparison with HMAC, ENVELOPE and AES to determine the basic performance of the algorithms. The HMAC and the hash functions of the SHA-512 algorithm are built-in in the GNUCrypto[III] API and were reused by the HX package. We implemented the padding and concatenations necessary for the ENVELOPE algorithm, before applying the SHA-512, also from GNUCrypto. The AES algorithm is available in the Java Cipher Class. For all the three algorithms, we used the same 256 bit Key. The GNUCrypto API is fully implemented in Java.

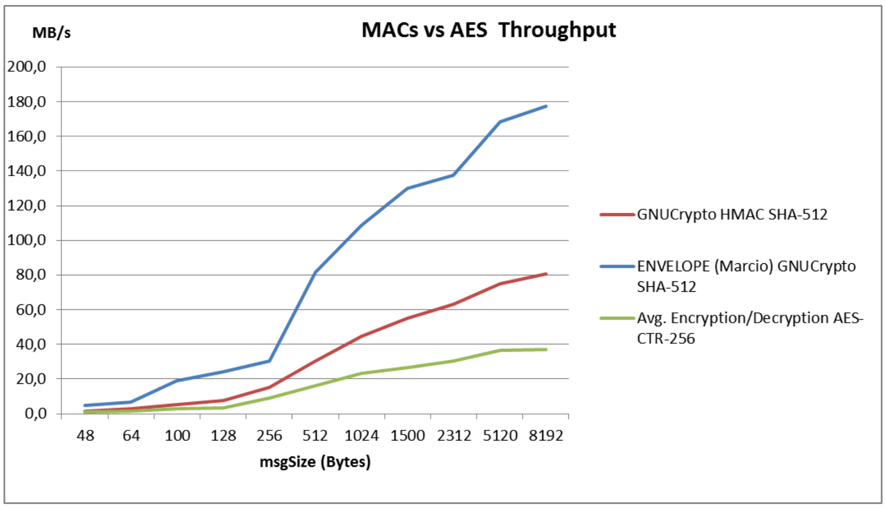


Figure 28 - MAC Algorithms vs AES Throughput

[III] The GNUCrypto Project - http://www.gnu.org/software/gnu-crypto/

The tests shows HMAC is twice as fast as AES and ENVELOPE is twice as fast as HMAC.

5.4.1 Encryption and Decryption Tests

In this scenario, we encrypted and decrypted messages of several sizes. We compared the time taken using AES-256-CTR against HX with most hash algorithms and cipher modes. We tested a JAVA application, executed in Core I7-2600 CPU, running at 3.40GHz with 16GB DDR3-1,333 RAM and 2 SSD Drives. The OS is Windows 7 Ultimate 64 bits. We obtained the following average encryption/decryption results:

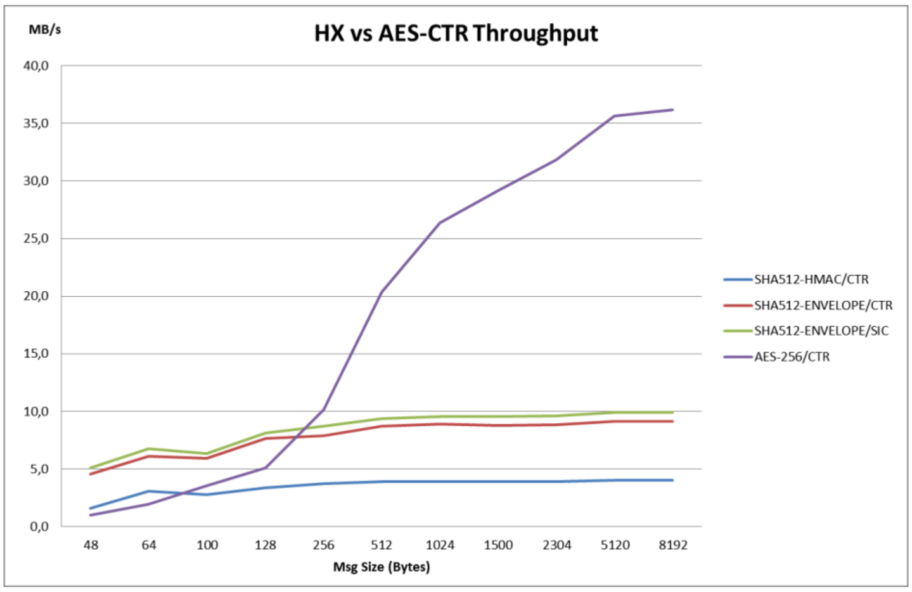
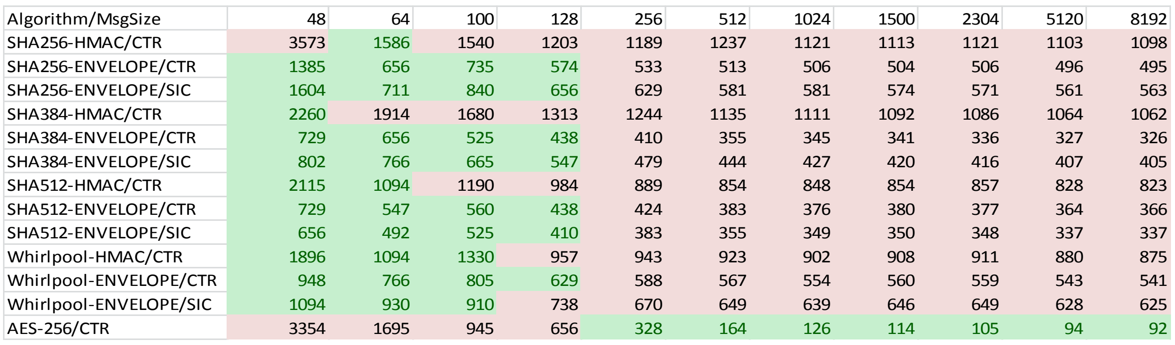


Figure 29 - HX vs AES Performance

The chart shows that for messages up to 200 Bytes, HX can beat AES. For messages greater than 200 Bytes, AES always beat HX. The greater the message size, the more AES outperforms HX. For 8KB messages, AES is twice as fast as HX. The hash algorithm which performed the best was SHA-512. With messages up to 512 Bytes it beats SHA-384. The latter beats the former, but the difference is not significant. However 512 bits of security instead of 384 bits is significant. Also a 512 bit keystream can encrypt 33% more information than a 384 bit [Table 6]. ENVELOPE beats HMAC in terms of speed, but the cost of manipulating salts in CTR operation mode made no difference against the single nonce of the SIC operation mode in HX. The table below shows the test results. We highlighted in green the cells where HX beets AES. The cells where HX had worse performance than AES were highlighted in red.



Algorithm/MsgSize	48	64	100	128	256	512	1024	1500	2304	5120	8192
SHA256-HMAC/CTR	3573	1586	1540	1203	1189	1237	1121	1113	1121	1103	1098
SHA256-ENVELOPE/CTR	1385	656	735	574	533	513	506	504	506	496	495
SHA256-ENVELOPE/SIC	1604	711	840	656	629	581	581	574	571	561	563
SHA384-HMAC/CTR	2260	1914	1680	1313	1244	1135	1111	1092	1086	1064	1062
SHA384-ENVELOPE/CTR	729	656	525	438	410	355	345	341	336	327	326
SHA384-ENVELOPE/SIC	802	766	665	547	479	444	427	420	416	407	405
SHA512-HMAC/CTR	2115	1094	1190	984	889	854	848	854	857	828	823
SHA512-ENVELOPE/CTR	729	547	560	438	424	383	376	380	377	364	366
SHA512-ENVELOPE/SIC	656	492	525	410	383	355	349	350	348	337	337
Whirlpool-HMAC/CTR	1896	1094	1330	957	943	923	902	908	911	880	875
Whirlpool-ENVELOPE/CTR	948	766	805	629	588	567	554	560	559	543	541
Whirlpool-ENVELOPE/SIC	1094	930	910	738	670	649	639	646	649	628	625
AES-256/CTR	3354	1695	945	656	328	164	126	114	105	94	92

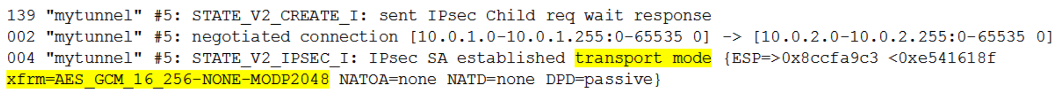
Table 30 - HX vs AES Performance Comparison (Cycles/Byte)

The experiment shows HX is fast enough to encrypt real time audio and video communications between two endpoints, since the encryption of an Ethernet frame (1500 Bytes) and a Wi-Fi Ethernet 802.11 frame (2304 Bytes) performs at hundreds of microseconds. Such an overhead is insignificant for real time audio and VoIP communications. VoIP quality is considered excellent when the delay time between endpoints is less than 150ms (Barry & Talha, 2013). With a more efficient bitwise XOR operation between the keystream and the plaintext and ciphered blocks concatenation, HX could beat AES-CTR in every scenario.

5.4.2 VPN Performance Tests

In this scenario, we assumed a client and a server had already negotiated a session key and wish to encrypt their communications with HX. We also compared HXBlock, setup to use SHA-512 ENVELOPE/SIC cipher mode, with AES-256-GCM (Galois Counter Mode) in a transport mode or host-host VPN.

A host-host VPN has a significantly lower overhead when compared with a tunnel-mode VPN, where the original packet is encapsulated by another set of IP headers. The transport mode, however, encrypts only the payload and ESP (Encapsulated Security Payload) trailer; hence the IP header of the original packet is not encrypted (Juniper Networks, 2017).



```
139 "mytunnel" #5: STATE_V2_CREATE_I: sent IPsec Child req wait response
002 "mytunnel" #5: negotiated connection [10.0.1.0-10.0.1.255:0-65535 0] -> [10.0.2.0-10.0.2.255:0-65535 0]
004 "mytunnel" #5: STATE_V2_IPSEC_I: IPsec SA established transport mode {ESP=>0x8ccfa9c3 <0xe541618f
xfrm=AES_GCM_16_256-NONE-MODP2048 NATOA=none NATD=none DPD=passive}
```

Figure 30 – VPN connection in transport mode

The environment consists of two virtual machines (VMs) in two different physical machines and a physical switch.

The test consists in downloading a text file from the server VM to the client VM. The file was generated by a real life banking application. The goal is to compare the transfer speed of the download operation. Each VM is located in a different physical machine: the physical computer and the physical server. The following figure better illustrates the test environment.

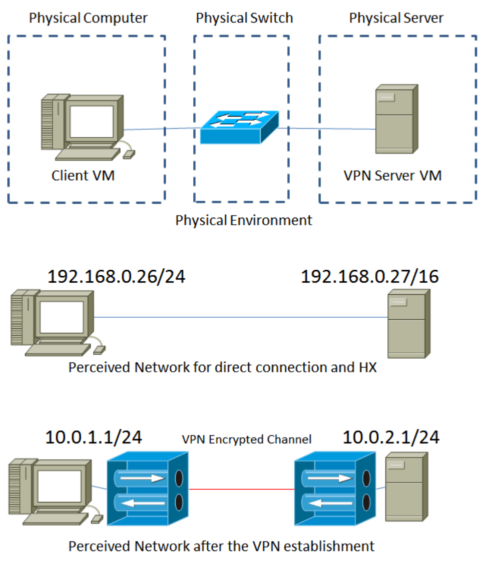


Figure 31 – VPN Test Environment

5.4.2.1 Hardware Description

The physical computer had an Intel Core I7-4770 CPU, running at 3.40GHz, with 16GB DDR3 1,600 RAM, a Gigabit NIC and 2 SSD Drives. The host OS was Windows 10 pro 64 bits.

The physical server had an AMD FX-8320 CPU, running at 3.5GHz, with 8GB DDR3 1,600 RAM, two Gigabit NIC and 2 SSD Drives. The host OS was the VMWare ESXi 6.7.0.

The physical switch was the LAN ports of a Draytek Vigor 2925 Router.

5.4.2.2 Virtual Machines Description

The client VM was configured with a RedHat Linux Enterprise 8.0 OS, 4GB non-shared memory, 2 VCPUs and one virtual NIC running in a VMWare Workstation 15.1.0. The VPN server VM was also configured with a RedHat Linux Enterprise 8.0, 4GB non-shared memory, 4 VCPUs and one virtual NIC. Network Description

The Client VM IP address was 192.168.0.26/24, while the server VM IP address was 192.168.0.27/24. The server application opened a TCP socket listening on the port 5501.

5.4.2.3 Tasks Description

We executed four tasks to test compare HX performance with AES. On all four tasks we transmit streams of 1500 Bytes (the stream buffer), which is the maximum size of an Ethernet frame. The file size was 11.3 Megabytes. The bandwidth was limited by the client VM, where we set both incoming and outgoing maximum transfer rates to 10, 20, 45 and 100 Mbps.

In the first task, we set the target IP address in the client VM application to 192.168.0.27. By doing so, the NAT on each router allows the client to connect to the server VM directly, without a VPN tunnel. Once the TCP connection was established, the server VM read 1500 Bytes from the file at a time and sent the stream to the client VM via the TCP socket. Upon receiving the stream, the client VM wrote the stream to the disk. The elapsed time was calculated and displayed in the console before the flushing operation and the closing of the downloaded file. The file was downloaded 20 times. The average transfer speed of this task, which represents the maximum performance of our test algorithm can reach, is our control variable.

In the second task, we established a transport mode VPN and set the target IP address in the client VM to 10.0.2.1. The rest of the task was exactly as described in the first task.

The third task is similar to the second task, but we established tunnel mode VPN instead by modifying the ipsec.conf file (RedHat Inc., 2019).

In the fourth task, the target IP address returned to 192.168.027. However, both the server and the client applications were modified to respectively encrypt and decrypt the stream, using HXBlock. The server encrypted the stream prior to sending it via the socket and the client decrypted the stream prior to writing it to the disk. By encrypting and decrypting the file, we simulated a VPN connection with HX as the encryption algorithm.

Both the client and the server codes were controlled by a single Boolean variable: usingEncryption. When the variable is set to false, encryption and decryption methods are not invoked. The average transfer speeds from tasks 2 to 4 were our dependent variables. We interfered neither in the MAC messages nor over the rekeying process, which occur in VPN communications.

The experiment produced the following results:

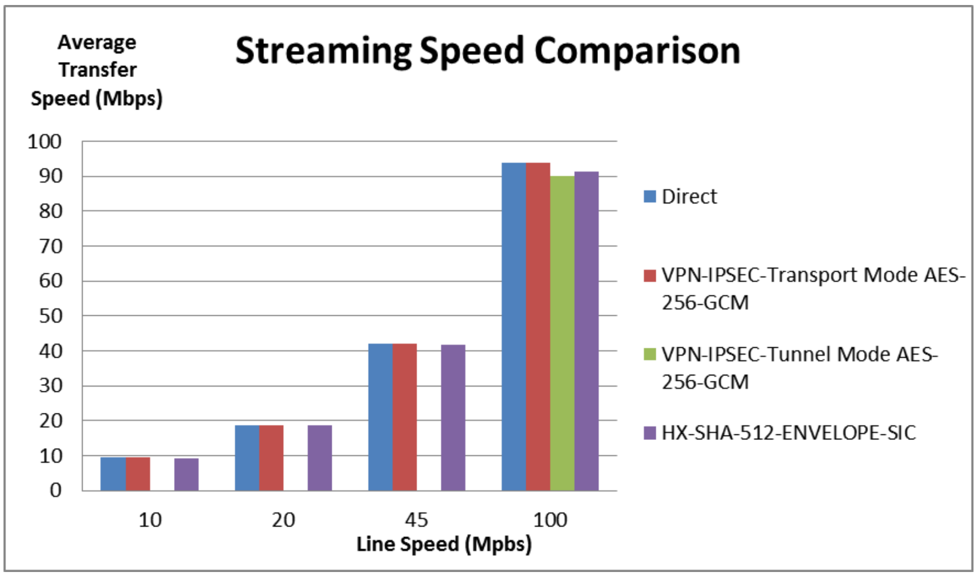


Figure 32 - HX vs AES VPN Throughput Comparison

The experiment shows the encryption cost is negligible. Considering the LibreSwan library, used by the IPSEC daemon in the RedHat Enterprise 8.0, is compiled in C and capable of calling AES hardware instructions, while HX is compiled in Java and there is no SHA-512 hardware instruction, we conclude both AES and HX algorithms achieved similar results in the experiment.

We can clearly see the encapsulation cost of the tunnel mode in the 100Mbps test. We were unable to conduct the test at lower speeds. The VPN tunnel fails and the IPSEC daemon had to be restarted on both virtual machines.

The performance tests show HX is very versatile and can be used in many scenarios to secure smart cities applications and encrypted data collection.

As an example, Cruz et al. proposed in 2010 an encryption scheme for exchanging SMS and MMS messages in a mobile network, which used AES-128 as the encryption algorithm (Cruz, et al., 2010). The scheme can have its security enhanced by HX, replacing AES-128, using keys up to 512 bits instead.

5.5 Comparison between HX and the State of the Art Algorithms

Based on the NIST key recommendations [Figure 1], the usability index in section [2.4.4.11] and the experiments from this section, we built the following tables for comparison of HX against the state of the art algorithms.

Algorithm	Block Cipher or Stream Cipher	Maximum Key Length	Usability Index
Twofish	Block	256	3
Serpent	Block	256	3
AES	Block	256	3
HC-256	Stream	256	7
Salsa20	Stream	256	7
HX	Stream	512	8

Table 31 - Encryption Algorithms Security and Usability Comparison

In Table 31], HX hash algorithm is either SHA-512 or Whirlpool. The maximum key length will be the same, if SHA3-512 is used or 1024 bits, if SKEIN-1024 is used. Since the hash algorithm and cipher mode are optional parameters, the only required parameters for HX is the plaintext, the encryption key, which is flexible and the function (encrypt or decrypt). This puts HX in the same Usability Index as the now unsafe RC4, with the advantage over the RC4 that HX produces different ciphered texts for the same key and plaintext each time the encryption method is invoked, while RC4 does not.

5.6 Post-Quantum Cryptography Comparison

Figure 1 shows the minimum key length recommendations for the near future, which applications for smart cities will have to comply with. The recommendations, however, do not consider Quantum Computing. If a Quantum Computer with enough qubits to brake current cyphers becomes available, post-quantum cryptography must be considered. Consequently, most algorithms in use today would not comply with minimum key length recommendations. The following table shows the impact of Quantum Computing resistance [2.8].

Algorithm	Maximum Key Length	Effective Key Length (Classical Computer)	Effective Key Length (Quantum Computer)	Complies with minimum key requirements
Twofish	256	255	128	No
Serpent	256	255	128	No
AES	256	255	128	No
HC-256	256	255	128	No
Salsa20	256	255	128	No
HX	512	511	256	Yes

Table 32 - Post-Quantum Cryptography Algorithm Comparison

6 Conclusions and Future Works

In this thesis, we proposed HX encryption algorithm and HX Authentication Protocol. Our experiments prove HX is safe, has a significant increase in key length, has the highest usability index among the state of the art algorithms and is modular in the sense that if a CHRF is broken or significantly wakened, the developer can choose another hash algorithm to generate keystreams without major changes in the code of the application.

Considering the fact the participants from the controlled experiment empirical study copied unsafe code from forums, the qualitative analysis of that experiment, the fact none of the participants were able to complete all tasks without any violations and the fact none of the participants from the survey experiment were able to answer all of the substantive questions correctly, we conclude the developers from the distinct samples lack the basic knowledge of cryptography concepts and are unskilled to develop an application that depends on cryptography to secure sensitive information. As a result, they will produce unsafe applications for the smart cities of the future.

Although the use of HX as the encryption algorithm might considerably mitigate usability problems, developers may still use a weak key for encryption. For that reason, developers of applications for smart cities should have mandatory formal training in cryptography

Encryption APIs could mitigate many encryption rules violations, if they have a “novice” default mode, which automatically instantiate the strongest algorithm, with the longest possible key, with a safe mode of operation, automatically creating a random IV, requiring only the key as a mandatory parameter, just like the HX API. “Expert” developers can decide if they want to change any of the default parameters.

HXAuth can be used with SRAP or other applications safely, easily implemented and with little overhead.

Developers and users of smart cities applications will benefit from HX and HXAuth, considering the strength of HX and that developers have fewer opportunities to violate cryptographic rules.

HX needs a more efficient bitwise XOR algorithm between keystreams and plaintexts to enhance encryption and decryption performance.

SHA-3-512 and SKEIN-1204 will be incorporated into the hash algorithms set to further enhance the strength of HX keystreams.

Glossary of Terms

$\oplus$	Bitwise XOR operation
$\|$	Concatenation operation
M	The plaintext message. The original unencrypted message
C	The ciphertext
M_i	The plaintext block indexed by i
C_i	The ciphertext block indexed by i
H(K,M)	The Message Authentication Code of the key K concatenated with the message M
H(i,K)	The hash of an integer i concatenated with a key K
E(K,M)	The encryption function E, using the key K to encrypt M
D(K,C)	The decryption function D, using the key K to decrypt C
KS_i	The keystream of the i block of a stream cipher
IV	Initialization Vector of an encryption algorithm or the Initial Value of a Hash
Φ_i	The ciphertext block indexed by I of the HX algorithm in CTR mode
S_i	The salt of ciphered block indexed by i of the HX algorithm in CTR mode
|H|	The length in bits of the digest produced by the hash algorithm H
|M|	The length in bits of the message M
π_K	The padding function of the key K
N	The nonce
C: Ops	The client of a client-server communications performs the operations Ops
S: Ops	The server of a client-server communications performs the operations Ops
$\rightarrow$S: Msg	The client sends the message Msg to the server
$\rightarrow$C: Msg	The server sends the message Msg to the client
x $\leftarrow$ cpt	The variable x receives the result of the computation cpt

Bibliographic References

Abidi, A. et al., 2016. Quantitative evaluation of chaotic CBC mode of operation. *Advanced Technologies for Signal and Image Processing (ATSIP)*, pp. 88-92.

Akhimullah, A; Hirose, S., 2016 . *Lightweight Hashing Using Lesamnta-LW Compression Function Mode and MDP Domain Extension.* Hiroshima, IEEE, pp. 590-596.

Allen, I; Seaman C, A., 2007. *Likert scales and data analyses.* [Online]
Available at: http://rube.asq.org/quality-progress/2007/07/statistics/likert-scales-and-data-analyses.html
[Accessed on 30 11 2017].

Anderson, R., 1999. *Serpent Home Page.* [Online]
Available at: http://www.cl.cam.ac.uk/~rja14/serpent.html
[Accessed on 18 06 2018].

Anil, J., Ross, A; Prabhakar, S., 2004. An Introduction to Biometric Recognition. *IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY*, 14(1), pp. 4-20.

Aoki, K. et al., 2009. Preimages for step-reduced SHA-2. In: *International Conference on the Theory and Application of Cryptology and Information Security.* Heidelberg - Germany: Springer, pp. 578-597.

Apple Support, 2017. *Safari and WebKit ending support for SHA-1 certificates.* [Online]
Available at: https://support.apple.com/HT207459
[Accessed on 21 06 2018].

ARM, 2015. *ARM Cortex-A57 MPCore Processor Cryptography Extension.* [Online]
Available at:
http://infocenter.arm.com/help/topic/com.arm.doc.ddi0514g/DDI0514G_cortex_a57_mpcore_cryptography_trm.pdf
[Accessed on 21 06 2018].

Asmara, R. J. A; Ardiansyah, R., 2017. Cipher feedback mode cryptographic algorithm with gingerbreadman two-dimensional map key generator on digital image. *IEEE Information & Communication Technology and System (ICTS)*, Issue 11, pp. 169-174.

Balebako, B. et. al., 2014. *The Privacy and Security Behaviors of Smartphone App Developers.* s.l., s.n.

Barker, E., 2016. *Recommendation for key Management.* [Online]
Available at: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-57pt1r4.pdf
[Accessed on 30 06 2018].

Barker, E., 2016. *Recommendation for Key Management.* [Online]
Available at: https://nvlpubs.nist.gov/nistpubs/specialpublications/nist.sp.800-57pt1r4.pdf
[Accessed on 02 04 2018].

Barreto, P; Rijmen, V., 2000. The Whirlpool hashing function. In: *First open NESSIE Workshop.* Leuven - Belgium: s.n., p. 14.

Barry, B; Talha, S., 2013. Evaluating the impact of AES encryption algorithm on Voice over Internet Protocol (VoIP) systems. *International Conference on Computing, Electrical and Electronics Engineering (ICCEEE),* Issue 1, pp. 686-691.

Bartkewitz, T., 2009. *Building hash functions from block ciphers, their security and implementation properties,* Bochum - Germany: Ruhr-University.

Basili, V; Weiss, D., 1984. A methodology for collecting valid software engineering data. *IEEE Trans. on Software Engineering,* SE-10(6), pp. 728-838.

Bellare, M., 2006. New proofs for NMAC and HMAC: Security without collision-resistance. *Annual International Cryptology Conference. Springer, Berlin, Heidelberg,* pp. 602-619.

Bellare, M., et. al., 1997. A concrete security treatment of symmetric encryption. In: *Proceedings 38th Annual Symposium on Foundations of Computer Science.* Miami Beach, FL, USA: IEEE, pp. 394-403.

Bellare, M., et. al., 2000. *A concrete Security Treatment of Symmetric Encryption,* San Diego, CA, USA: Dept. of Computer Sciense & Engineering, University of California San Diego.

Bellare, et. al., 2000. The security of the cipher block chaining message authentication code.. *Journal of Computer and System Sciences,* 61(3), pp. 362-399.

Bellare, et. al., 1998. Luby-Rackoff Backwards: Increasing Security by Making Block Ciphers Non-invertible. *Advances in Cryptology—Eurocrypt '98, Lecture Notes in Computer,* Volume 1402, pp. 266-280.

Bellare, M. *Key Distribution: PKI and Session-Key Exchange.* [Online]
Available at: http://cseweb.ucsd.edu/~mihir/cse207/s-kd.pdf
[Accessed on 07 08 2018].

Bellare, M; Rogaway, P., 2005. Block Ciphers. In: *Introduction to Modern Cryptography.* s.l.:s.n., pp. 39-40.

Bernstein, D., 2008. *ChaCha, a variant of Salsa20,* Chicago, USA: The University of Illinois at Chicago, Department of Mathematics, Statistics, and Computer Science (M/C 249).

Bernstein, D., 2008. The Salsa20 family of stream ciphers. In: *New stream cipher designs.* Heidelberg - Germany: Springer, pp. 84-97.

Bernstein, D., 2009. Is cryptography dead?. In: J. B. E. D. Daniel J. Bernstein, ed. *Post-Quantum Cryptography.* Darmstadt, Germany: Springer, p. 1.

Bertoni, G., et. al., 2007. Sponge functions. *ECRYPT hash workshop,* 2007(9).

Bertoni, G., et. al., 2009. *Keccak sponge function family main document. Submission to NIST (Round 2),* s.l.: s.n.

Bertoni, G., et. al., 2011. *Duplexing the Sponge: Single-Pass Authenticated Encryption and Other Applications.* Heidelberg - Germany, s.n.

Bertoni, G., et. al., 2013. Keccak. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Heidelberg - Germany: Springer, pp. 313-314.

Bhargavan, K; Leurent, G., 2016. On the practical (in-) security of 64-bit block ciphers: Collision attacks on HTTP over TLS and OpenVPN. In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security.* s.l.:ACM, pp. 456-467.

Biham, E., et. al., 2001. Linear cryptanalysis of reduced round Serpent. In: *International Workshop on Fast Software Encryption. Springer.* Heidelberg - Germany: Springer, pp. 16-27.

Biryukov, A., 2011. Related Key Attack. In: *Encyclopedia of Cryptography and Security.* Boston - USA: Springer, pp. 1040-1041.

Biryukov, A; Khovratovich, D., 2009. *Related-key Cryptanalysis of the Full AES-192 and AES-256,* Luxembourg: University of Luxembourg.

Bogdanov, A., et. al., 2011. Biclique cryptanalysis of the full AES. *Advances in Cryptology — ASIACRYPT,* 7073(Lecture Notes in Computer Science), pp. 344-371.

Boyd, C; Mathuria, A., 2003. A Tutorial Introduction to Authentication and Key Establishment. In: *Protocols for Authentication and Key Establishment.* Heidelberg - Germany: Springer-Verlag, pp. 23-31.

Braga, A; Dahab, R., 2017. *A Longitudinal and Retrospective Study on How Developers Misuse Cryptography in Online Communities..* Brasília, DF, Brazil., XVII Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais (SBSeg'17).

Braun, T., et. al., 2018. Security and privacy challenges in smart cities. *Sustainable Cities and Society v.39,* pp. 499 - 507.

Brose, G., 2014. Password. In: *Encyclopedia of cryptography and security.* s.l.:Springer Science & Business Media, pp. 453-455.

Campbell, K., et. al., 2003. *The economic cost of publicly announced information security breaches: empirical evidence from the stock market.* s.l., IOS Press, pp. 431-448.

Cheddad, A., et. al., 2010. *A hash-based image encryption algorithm.* s.l., Elsevier, pp. 879--893.

Chen, B., et. al., 2013. A Hybrid Mutual Identity Authentication Technology with its Application. *Computer security n.12*, pp. 34-37.

Cheng, Y., 2005. A New Text Digital Watermarking Algorithm. *Science & Technology and Engineering v.5 n.14*, pp. 1006-1008.

Chen, S; Jin, C., 2011. Distinguishing Attack on MAC with Enveloped Method Using MD5. *IEEE - International Conference on Network Computing and Information Security (NCIS),* Volume 1, pp. 48-51.

Chen, S., et. al., 2010. Side-channel leaks in web applications: A reality today, a challenge tomorrow. *IEEE Symposium on Security and Privacy,* pp. 191-206.

Claes, W. et al., 2000. *Experimentation in software engineering: an introduction.* s.l.:Springer Science & Business Media.

Cloutier, J; Vignesh, M., 2015. *Windows Enforcement of SHA1 Certificates.* [Online]
Available at:
https://social.technet.microsoft.com/wiki/contents/articles/32288.windows-enforcement-of-sha1-certificates.aspx
[Accessed on 21 06 2018].

Coron, J., et. al., 2005. *Merkle-Damgård Revisited: How to Construct a Hash Function.* Heidelberg - Germany, s.n.

Coskun, B; Memon, N., 2006. Confusion/diffusion capabilities of some robust hash functions. *IEEE 40th Annual Conference on Information Sciences and Systems*, 03, pp. 1188-1193.

Cruz, G, et. al., 2010. *IMPLEMENTAÇÃO DE CRIPTOGRAFIA APLICADA À COMUNICAÇÃO DE MENSAGENS SMS E MMS COM ESTEGANOGRAFIA EM TELEFONIA MÓVEL.* Algarve - Portugal, s.n.

Daemen, J; Kitsos, P., 2008. *The self-synchronizing stream cipher moustique, new stream cipher designs.* s.l., Springer, p. 210–223.

Dahab, R; López, J., 2007. O AES - Advanced Encryption Standard. In: *Técnicas criptográficas modernas: algoritmos e protocolos.* Campinas - Brazil: Instituto de Computação, pp. 16-20.

Dahab, R; López, J., 2007. Gerenciamento de Chaves. In: *Técnicas criptográficas modernas: algoritmos e protocolos.* Campinas - Brazil: Instituto de Computação, pp. 40-41.

Dahab, R; López, J., 2007. O DES - Data Enryption Standard. In: *Técnicas criptográficas modernas: algoritmos e protocolos.* Campinas - Brazil: Instituto de Computação , p. 13.

Dahab, R; López, J., 2007. Protocolos para Identificação Forte. In: *Técnicas criptográficas modernas: algoritmos e protocolos.* Campinas - Brazil: Instituto de Computação, p. 49.

Dahal, R., et. al., 2013. Performance Analysis of SHA-2 and SHA-3 finalists. *International Journal on Cryptography and Information Security (IJCIS),* 3(3), pp. 720-730.

Damgård, I., 1988. *Collision free hash functions and public key signature schemes.* s.l., D. Chaum and W.L. Price, Eds., Springer-Verlag, p. 203–216.

Damgård, I., 1989. A design principle for hash functions. *Springer - CRYPTO,* Volume 435 of LNCS, pp. 416-427.

de Magalhães, K., 2014. *Lattice-Based Predicate Encryption.* Doctoral dissertation, PhD thesis, University of Campinas ed. Campinas, SP, Brazil: UNICAMP.

Demirkan, H; Goul, M., 2013. *Taking value-networks to the cloud services: security services, semantics and service level agreements.* s.l., Springer, pp. 51-91.

Dinur, I., et. al., 2013. Collision attacks on up to 5 rounds of SHA-3 using generalized internal differentials. In: *International Workshop on Fast Software Encryption.* Heidelberg - Germany: Springer, pp. 219-240.

Dobbertin, H., et. al., 1996. RIPEMD-160: A strengthened version of RIPEMD. In: *FSE 1996. LNCS.* Heidelberg - Germany: Springer, pp. 71-82.

Dougherty, C., 2008. *"Vulnerability Note VU#836068 MD5 vulnerable to collision attacks",* Pittsburgh - USA: CERT Carnegie Mellon University Software Engineering Institute.

Ebrahim, M., et. al., 2013. Symmetric Algorithm Survey: A Comparative Analysis. *International Journal of Computer Applications,* 61(20).

Eckersley, P; Burns, J., 2010. *An Observatory for the SSLiverse.* [Online] Available at: https://www.eff.org/files/DefconSSLiverse.pdf
[Accessed on 16 07 2018].

Egele, M., et. al., 2013. *An empirical study of cryptographic misuse in android applications.* New York, NY, USA pp. 73-84, s.n., pp. 73-84.

El-Razouk, H., et. al., 2014. New Implementations of the WG Stream Cipher. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems V.22 N.9,* pp. 1865-1878.

Elzouka, H., 2006. *A New Robust and Secure Steganographic System for Greyscale Images,* Alexandria: Computer Engineering Department. Arab Academy for Science and Technology.

Elzouka, H. A. S. A., 2008. *FPGA based implementation of robust watermarking system.* s.l., IEEE, pp. 1274-1278.

Feistel, H., 05. Cryptography and Computer Privacy. *Scientific American*, 1973, pp. V 228, No 5, pp. 15-23.

Ferguson, N., 1999. Impossible differentials in Twofish. *Counterpane Systems,* Volume 19.

Ferguson, N., et al., 2010. *The Skein hash function family,* s.l.: Submission to NIST (round 3).

Ferguson, N., et. al., 2010. In: *Cryptography Engineering: Design Principles and Practical Applications.* Hoboken - USA: Wiley Publishing, pp. 63-64.

Feynman, R., 1982. Simulating Physics with Computers. *International Journal of Theoretical Physics*, pp. 467-488.

Fontaine, C., 2011. Self-Synchronizing Stream Cipher. *Encyclopedia of Cryptography and Security,* pp. 1175-1176.

Fontaine, C., 2011. Synchronous Stream Cipher. *Encyclopedia of Cryptography and Security,* pp. 1274-1275.

Giry, D; Quisquater, J., 2017. *BlueKrypt - Cryptographic Key Lenght Recomendation.* [Online]
Available at: https://www.keylength.com/en/compare/
[Accessed on 06 06 2018].

Google, 2014. *Intent to Deprecate SHA-1 certificates.* [Online]
Available at: https://groups.google.com/a/chromium.org/d/msg/blink-dev/2-R4XziFc7A/i_JipRRJoDQJ
[Accessed on 21 06 2018].

Gordon, L. A; Loeb, M. P., 2001. *Using information security as a response to competitor analysis systems.* s.l., ACM, pp. 70--75.

Gordon, L. A; Loeb, M. P., 2004. The economics of information security investment. In: *Economics of Information Security.* s.l.:Springer, pp. 105-125.

Gordon, L; Loeb, M., 2002. *The economics of information security investment.* s.l., ACM, pp. 438-457.

Grisenthwaite, R., s.d. *ARMv8 Technology Preview.* [Online]
Available at: https://www.arm.com/files/downloads/ARMv8_Architecture.pdf
[Accessed on 19 02 2019].

Grover, L., 1996. A fast quantum mechanical algorithm for database search. *Proceedings, 28th Annual ACM Symposium on the Theory of Computing*, p. 212.

Gunther, C. G., 1990. An identity-based key-exchange protocol. *Advances in Cryptology EUROCRYPT*, LNCS 434(89), p. 29–37.

Guo, J., et. al., 2013. Cryptanalysis of HMAC/NMAC-whirlpool. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Heidelberg - Germany: Springer, pp. 21-40.

Håstad, J., et. al., 1999. A Pseudorandom Generator from any One-way Function. *SIAM Journal on Computing*, 28(4), pp. 1364-1396.

Heys, H., 2002. A Tutorial on Linear and Differential Cryptanalysis. *Cryptologia*, 26(3), pp. 189-221.

Huang, K., Chiu, J; Shen, S., 2013. A Novel Structure with Dynamic Operation Mode for Symmetric-Key Block Ciphers. *International Journal of Network Security & Its Applications*, 5(1), p. 17.

Huang, Z., et. al., 2001. Hash - based Encryption Scheme. *Communications Technology n.7*, pp. 87-89.

IBM Crypto Development Team, 2015. *Introduction to IBM z Systems Cryptography and the Ecosystem around z Systems Cryptography.* [Online]
Available at:
https://www.ibm.com/developerworks/community/files/form/anonymous/api/library/77ca416f-4e0e-4689-83a1-22a30b828c26/document/9a5a15b8-8cc1-485d-b64f-f4f1379d9440/media
[Accessed on 19 02 2018].

Intel Corporation, 2013. *Intel® SHA Extensions.* [Online]
Available at: https://software.intel.com/en-us/articles/intel-sha-extensions
[Accessed on 21 06 2018].

Intel Corporation, s.d. *Intel® Core™ i5-6400T Processor.* [Online]
Available at:
https://www.intel.com/content/www/us/en/products/processors/core/i5-processors/i5-6400t.html
[Accessed on 19 02 2018].

Jin, Z. et al., 2017. A New and Practical Design of Cancellable Biometrics: Index-of-Max Hashing. *arXiv preprint arXiv:1703.05455.*

Joux, A., 2004. Multicollisions in iterated hash functions. Application to cascaded constructions. In: *Annual International Cryptology Conference.* Heidelberg - Germany: Springer, pp. 306-316.

Jueneman, R., 1983. Analysis of certain aspects of output feedback mode. In: *Advances in Cryptology.* Boston - USA: Springer, pp. 99-127.

Juniper Networks, 2017. *What is the difference between the Tunnel and Transport modes in ESP?.* [Online]
Available at:
https://kb.juniper.net/InfoCenter/index?page=content&id=KB5302&actp=METADATA
[Accessed on 04 06 2019].

Kahn, D., 1967. The Codebreakers: The Story of Secret Writing. *New York: Macmillan Publishing Co..*

Kam, J; Davida, G., 1979. Structured design of substitution-permutation encryption networks. *IEEE Transactions on Computers,* Volume 10, pp. 747-753.

Kasunic, M., 2005. *Designing an effective survey,* Pittsburgh - USA: Carnegie-Mellon Univeristy Software Engineering Institute.

Kelsey, J; Schneier, B., 2005. *Second preimages on n-bit hash functions for much less than 2n work.* s.l., Springer, p. 474–490.

Kerchhoff, A., 1883. *La Cryptographie Militaire.* [Online]
Available at: http://www.petitcolas.net/kerckhoffs/crypto_militaire_1.pdf
[Accessed on 11 06 2016].

Khovratovich, D., et. al., C., 2012. Narrow-Bicliques: cryptanalysis of full IDEA. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Heidelberg - Germany: Springer, pp. 392-410.

Khovratovich, D., Nikolić, I; C., R., 2010. Rotational rebound attacks on reduced Skein. In: *nternational Conference on the Theory and Application of Cryptology and Information Security.* Heidelberg - Germany: Springer, pp. 1-19.

Kim, J., et. al., 2006. On the security of HMAC and NMAC based on HAVAL, MD4, MD5, SHA-0 and SHA-1. In: *International Conference on Security and Cryptography for Networks.* Heidelberg - Germany: Springer, pp. 242-256.

Krawczyk, H., et. al., 1997. *RFC-2104: HMAC: Keyed-Hashing for Message Authentication,* s.l.: IETF.

Kumari, S., et. al., 2012. *Cryptanalysis and security enhancement of Chen et al.'s remote user authentication scheme using smart card.* s.l., Springer, pp. 60-75.

Kumari, S; Khan, M., 2014. *Cryptanalysis and improvement of 'a robust smart-card-based remote user password authentication scheme'.* s.l., Wiley Online Library, pp. 3939-3955.

Lai, X; Massey, J., 1990. A proposal for a new block encryption standard. In: *Workshop on the Theory and Application of of Cryptographic Techniques.* Berlin - Germany: Springer, pp. 389-404.

Lardinois, F., 2019. *IBM unveils its first commercial quantum computer.* [Online]
Available at: https://techcrunch.com/2019/01/08/ibm-unveils-its-first-commercial-quantum-computer/
[Accessed on 05 06 2019].

Lazar, D. et. al., 2014. *Why does cryptographic software fail? A case study and open problems.* s.l., s.n.

Lee, C., et. al., 2006. *Security enhancement on a new authentication scheme with anonymity for wireless environments.* s.l., IEEE, pp. 1683-1687.

Lenstra, A; Verheul, E., 2001. Selecting Cryptographic Key Sizes. In: *Journal of Cryptology*. s.l.:International Association for Cryptologic Research, pp. 255-293.

Linaker, J. et al., 2015. *Guidelines for conducting surveys in software engineering,* Rio de Janeiro - Brazil: COPPE-RJ.

Lipmaa, H., et. al., 2000. *Comments to NIST concerning AES Modes of Operations: CTR-Mode Encryption,* Berkeley - USA: University of California Berkeley.

Li, S., et. al., 2003. A New Message Digest Codes Generating Algorithm. *Journal of Computer Research and Development v.40 n.3,* pp. 413-416.

Luby, M; Rackoff, C., 1988. How to Construct Pseudorandom Permutations from Pseudorandom Functions. *SIAM Journal on Computing,* 17(2), pp. 373-386.

Lui, H., 2018. *What Is the Future of Ecommerce in 2018 and Beyond? 10 Trends.* [Online]
Available at: https://www.shopify.com/enterprise/the-future-of-ecommerce
[Accessed on 05 06 2018].

Mahfouf, M., et. al., 2002. Fuzzy model-based predictive control using an ARX structure with feedforward. In: *Fuzzy sets and systems.* s.l.:s.n., pp. 39-59.

Mahon, A., 2003-2007. *THE HISTORY OF HUT EIGHT.* [Online]
Available at: http://www.ellsbury.com/hut8/hut8-000.htm
[Accessed on 29 06 2018].

Manger, J., 2001 pp. 230-238. *A chosen ciphertext attack on RSA optimal asymmetric encryption padding (OAEP) as standardized in PKCS# 1 v2.0.* Heidelberg - Germany, s.n.

McKinney, E. H., 1966. Generalized birthday problem. *The American Mathematical Monthly v. 73 n. 4,,* pp. 385-387.

McLaren, D; Agyeman, J., 2015. *Sharing Cities: A Case for Truly Smart and Sustainable Cities.* Massachusetts - USA: MIT Press.

Mendel, F., et. al., 2009. he rebound attack: Cryptanalysis of reduced Whirlpool and Grøstl. In: *Fast Software Encryption.* Heidelberg - Germany: Springer, pp. 260-276.

Merkle, R., 1989. One way hash functions and DES. *Springer CRYPTO,* Volume 435 of LNCS, p. 428–446.

Metzge, P; Simpson, W., 1995. *IP authentication using keyed MD5,* s.l.: IETF, RFC 1828.

Metzge, P; Simpson, W., 1995. *IP authentication using keyed SHA,* s.l.: IETF, RFC 1852.

Mitchell, J., et. al., 1998. Finite-State Analysis of SSL 3.0. *USENIX Security Symposium*, 01, pp. 201-216.

Mitchell, K; Kim, E., 2017. *AMD RYZEN™ CPU OPTIMIZATION.* [Online]
Available at: http://32ipi028l5q82yhj72224m8j.wpengine.netdna-cdn.com/wp-content/uploads/2017/03/GDC2017-Optimizing-For-AMD-Ryzen.pdf
[Accessed on 21 06 2018].

Morris, C., 1993. Navy Ultra's Poor Relations. In: *Codebreakers: The inside story of Bletchley Park.* Oxford: Oxford university Press, p. 235.

Mouha N, P. B., 2013. *Towards finding optimal differential characteristics for ARX: Application to Salsa20.,* s.l.: Cryptology ePrint Archive, Report 2013/328.

Mozilla Security Blog, 2014. *"Phasing Out Certificates with SHA-1 based Signature Algorithms.* [Online]
Available at: https://blog.mozilla.org/security/2014/09/23/phasing-out-certificates-with-sha-1-based-signature-algorithms/
[Accessed on 21 06 2018].

Musa, S., 2016. *Smart City Roadmap.* [Online]
Available at: https://www.academia.edu/21181336/Smart_City_Roadmap
[Accessed on 05 06 2018].

National Security Agancy/Central Security Service/Information Assurence Directorate, 2016. *Commercial National Security Algorithm Suite and Quantum Computing FAQ.* [Online]
Available at: https://cryptome.org/2016/01/CNSA-Suite-and-Quantum-Computing-FAQ.pdf
[Accessed on 06 06 2018].

NIST - National Institute of Standards and Technology, 2015. *Secure Hash Standard (SHS).* [Online]
Available at:
https://csrc.nist.gov/csrc/media/publications/fips/180/4/final/documents/fips180-4-draft-aug2014.pdf
[Accessed on 21 06 2018].

NIST Computer Security Division's (CSD) Security Technology Group (STG), 2012. *Block Cipher Modes,* Gaithersburg - USA: NIST.

NSA - National Security Agency, 2016. *Algorithms to Support the Evolution of Information Assurance Needs.* [Online]
Available at:
https://www.iad.gov/iad/customcf/openAttachment.cfm?FilePath=/iad/library/ia-guidance/ia-solutions-for-classified/algorithm-guidance/assets/public/upload/Algorithms-to-Support-the-Evolution-of-Information-Assurance-Needs.pdf&WpKes=aF6woL7fQp3dJieMru6YSQnL
[Accessed on 02 05 2018].

Ohtahara, C., et. al., 2010. Preimage attacks on step-reduced RIPEMD-128 and RIPEMD-160. In: *International Conference on Information Security and Cryptology*. Heidelberg - Germany: Springer, pp. 169-186.

Pascal, J., 2001. *On the Complexity of Matsui's Attack,* Lausanne - Swiss: Swiss Institute of Technology.

Patrick, K. N., 2008. *Method of comparing documents possessed by two parties.* USA, Patente Nº 7,337,319.

Peyravian, M., et. al., 1999. *Hash-based encryption system.* s.l., Elsevier, pp. 345-350.

Preneel, B., 2003. *Analysis and Design of Cryptographic Hash Functions.* s.l.:Diss. PhD thesis, Katholieke Universiteit Leuven. pp. 18.

Preneel, B., 2003. Applications of HASH Functions. In: *Analysis and design of cryptographic hash functions.* Leuven - Nethherlands.: Doctoral dissertation, Katholieke Universiteit te Leuven, p. 19.

Preneel, B., 2007. An Introduction to Modern Cryptography. *The History of Information Security: A Comprehensive Handbook,* pp. 565-592.

Preneel, B., et. al., 1998. *Principles and Performance of Cryptographic Algorithms.* [Online]
Available at: http://www.drdobbs.com/algorithm-alley/184410756
[Accessed on 12 06 2018].

Rayan, A. M., et. al., 2016. Provably Secure Encryption Algorithm based on Feistel Structure. *International Journal of Computer Applications,* Volume I, p. 139.

RedHat Inc., 2019. *Securing Virtual Private Networks (VPNs) Using Libreswan.* [Online]
Available at: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html/security_guide/sec-securing_virtual_private_networks
[Accessed on 04 06 2019].

Rijman, V., 1997. *Cryptanalysis and design of iterated block ciphers.* Leuven - Belgium: Doctoral Dissertation.

Rogaway, P., 2002. *Nonce-Based Symmetric Encryption,* Davis - USA: Dept. of Computer Science, University of California.

Rogaway, P; Shrimpton, T., 2004. *Cryptographic hash-function basics: Definitions, implications, and separations for preimage resistance, second-preimage resistance, and collision resistance.* Berlin-Heidelberg, Springer, pp. 371-388.

ROSEMBERG, M., 2014. *SRAP - A new Authentication Protocol for Semantic Web Applications MsC. Thesis.* Rio de Janeiro. Retrieved from:

http://www.dbd.puc-rio.br/pergamum/tesesabertas/1221733_2014_completo.pdf: Pontifícia Universidade Católica do Rio de Janeiro, Dep. of Informatics.

Rueppel, R. A., 1986. *"Stream ciphers." Analysis and Design of Stream Ciphers.*. Berlin Heidelberg: Springer pp 5-16..

Sasaki, Y; Aoki, K., 2009. Finding preimages in full MD5 faster than exhaustive search. In: *nnual International Conference on the Theory and Applications of Cryptographic Techniques.* Heidelberg - Germany: Springer, pp. 134-152.

Schneier, B., 1993. *Description of a New Variable-Length Key, 64-Bit Block Cipher (Blowfish).* [Online]
Available at:
https://www.schneier.com/academic/archives/1994/09/description_of_a_new.html
[Accessed on 18 06 2018].

Schneier, B., 1996. Counter Mode. In: *Applied cryptography: protocols, algorithms, and source code in C (Second Edition).* New York: John Wiley & Sons, Inc., pp. 178-179.

Schneier, B., 1996. Using one-Way Hash Functions. In: *Applied cryptography: protocols, algorithms, and source code in C (Second Edition).* New York: John Wiley & Sons, Inc., p. 296.

Schneier, B. et al., 2000. *The Twofish Team's Final Comments on AES.* [Online]
Available at: https://www.schneier.com/academic/paperfiles/paper-twofish-final.pdf
[Accessed on 18 06 2018].

Schwartz, M., 2011. *Stolen Digital Certificates Compromised CIA, MI6, Tor.* [Online]
Available at: http://www.darkreading.com/attacks-and-breaches/stolen-digital-certificates-compromised-cia-mi6-tor/d/d-id/1099964?
[Accessed on 24 07 2018].

Sekar G, P. B., 2009. Improved distinguishing attacks on HC-256. In: *International Workshop on Security.* Heidelberg - Germany: Springer, pp. 38-52.

Shahzad, R., 2012. *Lecture3a symmetric encryption.* [Online]
Available at: https://www.slideshare.net/rajakhurram/lecture3a-symmetric-encryption02
[Accessed on 12 06 2018].

Shannon, C., 1949. Communications Theory of Secrecy Systems. *Bell System Thecnical Journal,* 28(4), pp. 656-715.

Shor, P., 1997. "Polynomial-Time Algorithms for Prime Factorization and Discrete Logarithms on a Quantum Computer". *SIAM Journal on Computing,* 26(5), p. 1484–1509.

Shrimpton, T; Stam, M., 2008. Building a collision-resistant compression function from non-compressing primitives. In: *International Colloquium on Automata, Languages, and Programming.* Heidelberg - Germany: Springer, pp. 643-654.

Stallings, W., 2011. Block Cipher Operation. In: *Cryptography and Network Security Principles and Practice Fifth Edition.* s.l.:Prentice Hall, pp. 196-197.

Stallings, W., 2011. Classical Encryption Techniques. In: *Cryptography and Network Security Principles and Practice Fifth Edition.* New York - USA: Prentice-Hall, pp. 33-35.

Stallings, W., 2011. Cryptographic Hash Functions. In: *Cryptography and Network Security Principles and Practice Fifth Edition.* New York - USA: Prentice-Hall, pp. 340-342.

Stallings, W., 2011. Public-Key Cryptography and Message Authentication. In: *Network Security Essentials: Applications and Standards (4th edition).* Boston - USA: Prentice Hall, pp. 78-79.

Stallings, W., 2011. RC4. In: *Network Security Essentials: Applications and Standards (4th edition).* Boston - USA: Prentice Hall, pp. 234-236.

Stallings, W., 2011. SYMMETRIC CIPHER MODEL. In: *Cryptography and Network Security Principles and Practice Fifth Edition.* s.l.:Prentice Hall, p. 33.

Stamp, M; Low, R., 2007. FEAL-4 Differential Attack. In: *Applied Cryptanalysis: breaking Ciphers in the Real World.* Hoboken - USA: Wiley & Sons, pp. 170-177.

Stamp, M; Low, R., 2007. FEAL-4 Linear Attack. In: *Applied Cryptanalysis: breaking Ciphers in the Real World.* Hoboken - USA: Wiley & Sons, pp. 177-182.

Stamp, M; Low, R., 2007. Introduction. In: *Applied Cryptanalysis: breaking Ciphers in the Real World.* Hoboken - USA: Wiley & Sons, pp. 1-4.

Stevens, M. et al., 2017. *The first collision for full SHA-1,* Amsterdam - Netherlands: CWI Amsterdam - Google Research.

Su, S., et. al., 2016. A provably secure non-iterative hash function resisting birthday attack. In: *Theoretical Computer Science v.654.* s.l.:Elsevier, pp. 128-142.

Tesink, S., et. al., 2005. *Improving csirt communication through standardized and secured information exchange,* s.l.: Tilburg Master Thesis.

Tews, E., Weinmann, R; Pyshkin, A., 2007. *Breaking 104 Bit WEP in Less Than 60 Seconds,* s.l.: Cryptology ePrint Archive, Report 2007/120.

The Ministry of Internal Affairs and Communication of Japan and The Ministry of Economy, Trade and Industry of Japan, 2003. *e-Government recommended ciphers list.* [Online]

Available at: http://www.cryptrec.go.jp/english/images/cryptrec_01en.pdf
[Accessed on 21 06 2018].

Tsudik, G., 1992. Message authentication with one-way hash functions. In: *INFOCOM'92. Eleventh Annual Joint Conference of the IEEE Computer and Communications Societies.* s.l.:IEEE, pp. 2055-2059.

Tsunoo Y, et. al., 2007. Differential cryptanalysis of Salsa20/8. *Workshop Record of SASC 2007 Vol. 28.*

Turner, S; Chen, L., 2011. *RFC-6151: Updated Security Considerations for the MD5 Message-Digest and the HMAC-MD5 Algorithms,* s.l.: IETF.

Vanhoef, M; Piessens, F., 2015. *All Your Biases Belong To Us: Breaking RC4 in WPA-TKIP and TLS,* Washington DC - USA: KU Leuven.

Vernam, G., 1926. Cipher printing telegraph system for secret wire and radio telegraph communications. *Journal American Institute of Electrical Engineers Vol. XLV,* pp. 109-115.

Votipka, D. et al., 2019. *Understanding security mistakes developers make: Qualitative analysis from Build It, Break It, Fix It,* College Park, MD, USA: University of Maryland.

Vũ, H., 2012. *MD5 Length Extension Attack Revisited.* [Online]
Available at: http://vudang.com/2012/03/md5-length-extension-attack/
[Accessed on 27 06 2018].

Wang, R., et. al., 2011. *Robust authentication and key agreement scheme preserving the privacy of secret key.* s.l., Elsevier, pp. 274-280.

Wang, X. et al., 2009. Cryptanalysis on hmac/nmac-md5 and md5-mac. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Heidelberg - Germany: Springer, pp. 121-133.

Wu, H., 2004. A New Stream Cipher HC-256. In: *International Workshop on Fast Software Encryption.* Heidelberg - Germany: Springer, pp. 226-244.

Xian, L; Tingthanathikul, W., 2004. *Advanced Encryption Standard (AES) in Counter Mode,* s.l.: s.n.

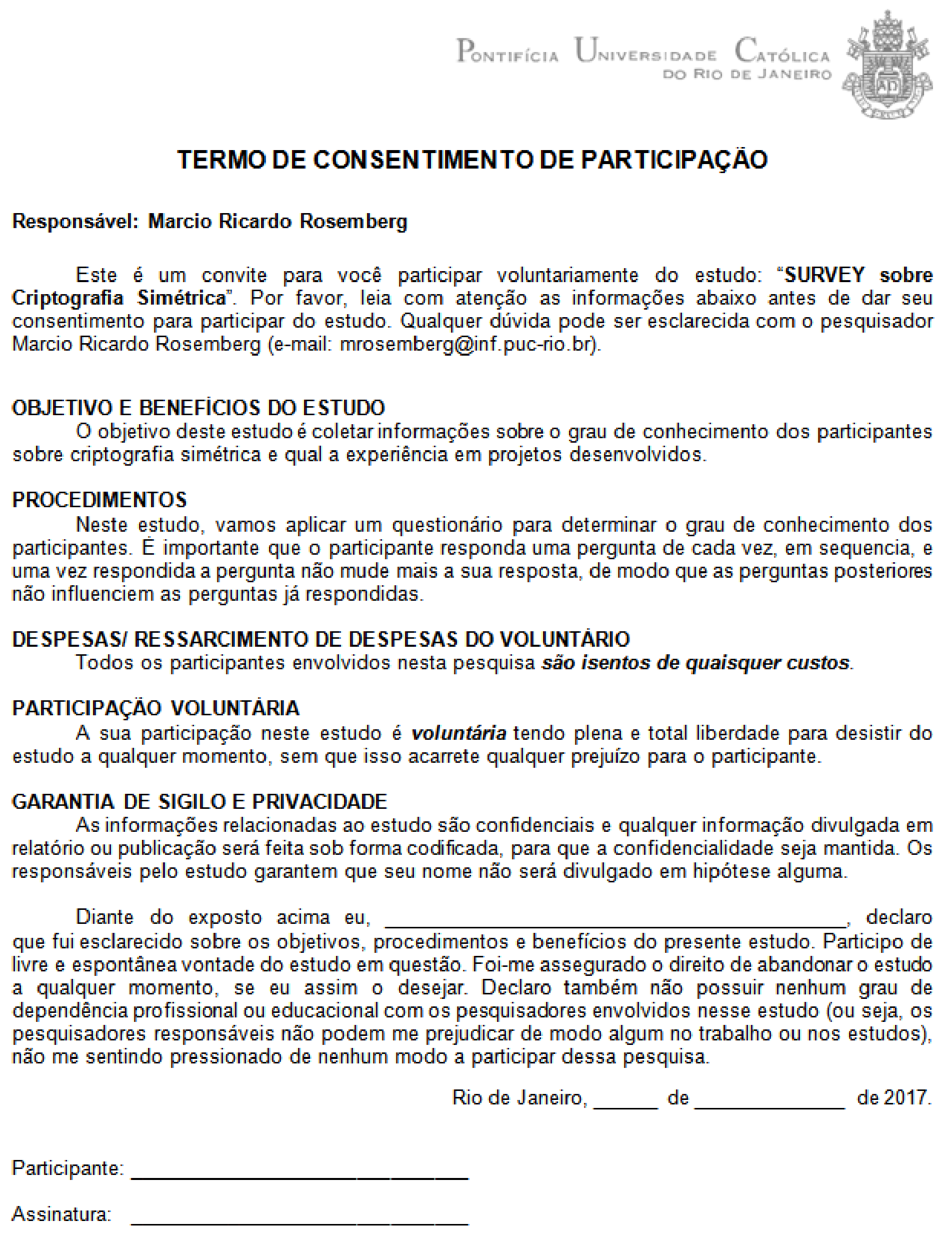
Xie, T., et. al., 2013. *Fast Collision Attack on MD5.* [Online]
Available at: https://eprint.iacr.org/2013/170.pdf
[Accessed on 21 06 2018].

Yeh, Y., et. al., 2001. *RC hash function.* s.l., Taylor & Francis, pp. 297-306.

Zhou, Y; Feng, D., 2005. Side-Channel Attacks: Ten Years After Its Publication and the Impacts on Cryptographic Module Security Testing.. *IACR Cryptology ePrint Archive,* p. 388.

Zuccherato, R., 2014. Authentication Token. In: *Encyclopedia of cryptography and security*. s.l.:Springer Science & Business Media, pp. 23-24.

Annex 1 Survey Participants Consent Form



Pontifícia Universidade Católica do Rio de Janeiro

TERMO DE CONSENTIMENTO DE PARTICIPAÇÃO

Responsável: Marcio Ricardo Rosemberg

Este é um convite para você participar voluntariamente do estudo: "**SURVEY sobre Criptografia Simétrica**". Por favor, leia com atenção as informações abaixo antes de dar seu consentimento para participar do estudo. Qualquer dúvida pode ser esclarecida com o pesquisador Marcio Ricardo Rosemberg (e-mail: mrosemberg@inf.puc-rio.br).

OBJETIVO E BENEFÍCIOS DO ESTUDO
O objetivo deste estudo é coletar informações sobre o grau de conhecimento dos participantes sobre criptografia simétrica e qual a experiência em projetos desenvolvidos.

PROCEDIMENTOS
Neste estudo, vamos aplicar um questionário para determinar o grau de conhecimento dos participantes. É importante que o participante responda uma pergunta de cada vez, em sequencia, e uma vez respondida a pergunta não mude mais a sua resposta, de modo que as perguntas posteriores não influenciem as perguntas já respondidas.

DESPESAS/ RESSARCIMENTO DE DESPESAS DO VOLUNTÁRIO
Todos os participantes envolvidos nesta pesquisa ***são isentos de quaisquer custos***.

PARTICIPAÇÃO VOLUNTÁRIA
A sua participação neste estudo é ***voluntária*** tendo plena e total liberdade para desistir do estudo a qualquer momento, sem que isso acarrete qualquer prejuízo para o participante.

GARANTIA DE SIGILO E PRIVACIDADE
As informações relacionadas ao estudo são confidenciais e qualquer informação divulgada em relatório ou publicação será feita sob forma codificada, para que a confidencialidade seja mantida. Os responsáveis pelo estudo garantem que seu nome não será divulgado em hipótese alguma.

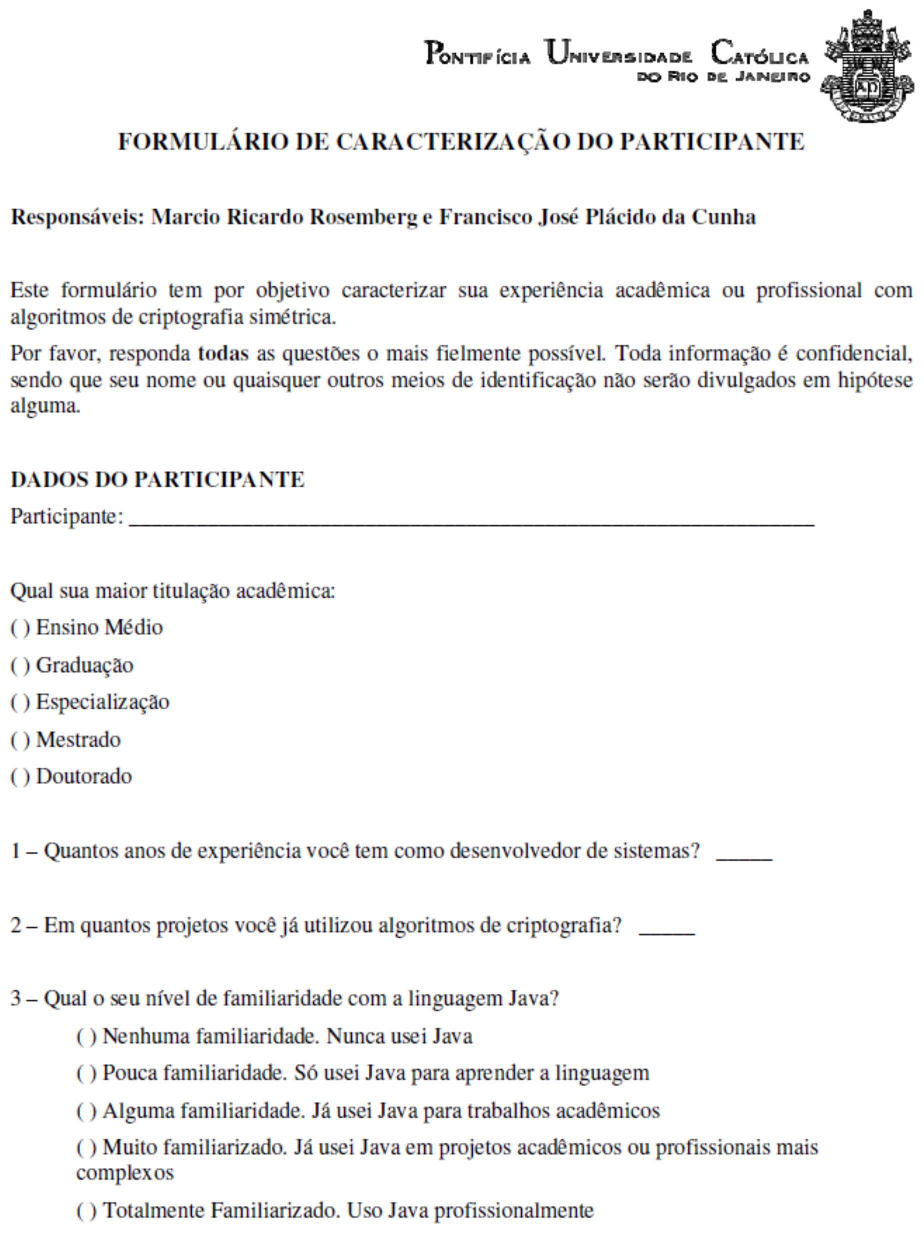
Diante do exposto acima eu, ____, declaro que fui esclarecido sobre os objetivos, procedimentos e benefícios do presente estudo. Participo de livre e espontânea vontade do estudo em questão. Foi-me assegurado o direito de abandonar o estudo a qualquer momento, se eu assim o desejar. Declaro também não possuir nenhum grau de dependência profissional ou educacional com os pesquisadores envolvidos nesse estudo (ou seja, os pesquisadores responsáveis não podem me prejudicar de modo algum no trabalho ou nos estudos), não me sentindo pressionado de nenhum modo a participar dessa pesquisa.

Rio de Janeiro, ______ de ______________ de 2017.

Participante: ________________________________

Assinatura: ________________________________

Annex 2 Controlled Experiment Subject Characterization Form



Pontifícia Universidade Católica do Rio de Janeiro

FORMULÁRIO DE CARACTERIZAÇÃO DO PARTICIPANTE

Responsáveis: Marcio Ricardo Rosemberg e Francisco José Plácido da Cunha

Este formulário tem por objetivo caracterizar sua experiência acadêmica ou profissional com algoritmos de criptografia simétrica.

Por favor, responda **todas** as questões o mais fielmente possível. Toda informação é confidencial, sendo que seu nome ou quaisquer outros meios de identificação não serão divulgados em hipótese alguma.

DADOS DO PARTICIPANTE

Participante: ____

Qual sua maior titulação acadêmica:

( ) Ensino Médio

( ) Graduação

( ) Especialização

( ) Mestrado

( ) Doutorado

1 – Quantos anos de experiência você tem como desenvolvedor de sistemas? _____

2 – Em quantos projetos você já utilizou algoritmos de criptografia? _____

3 – Qual o seu nível de familiaridade com a linguagem Java?

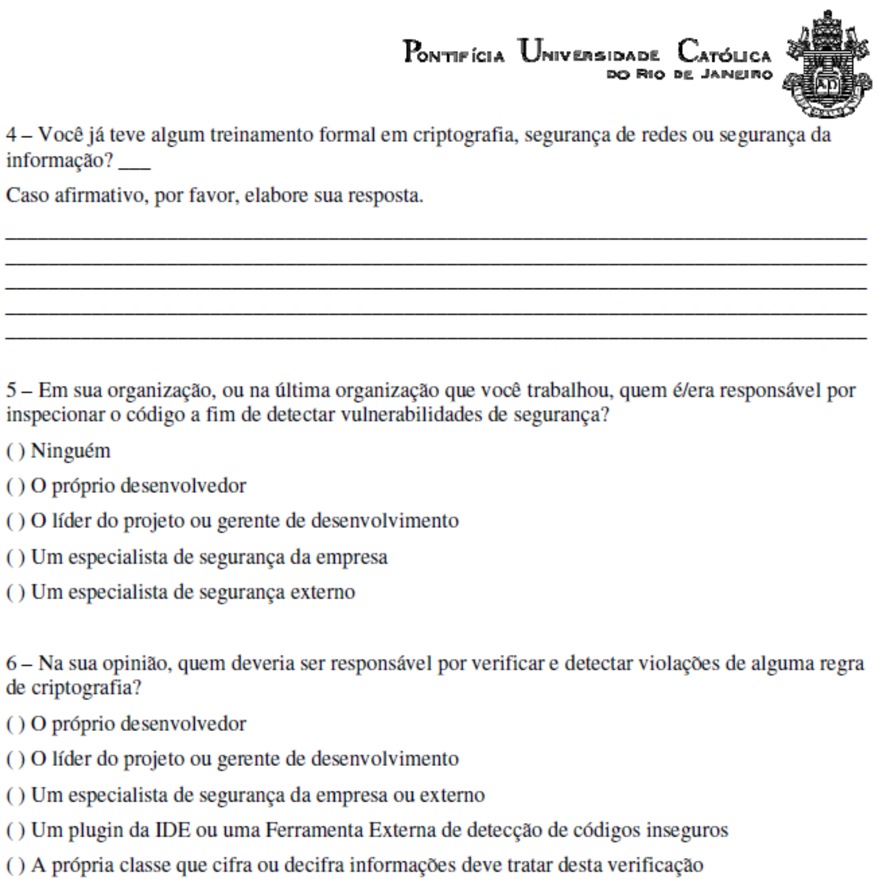
( ) Nenhuma familiaridade. Nunca usei Java

( ) Pouca familiaridade. Só usei Java para aprender a linguagem

( ) Alguma familiaridade. Já usei Java para trabalhos acadêmicos

( ) Muito familiarizado. Já usei Java em projetos acadêmicos ou profissionais mais complexos

( ) Totalmente Familiarizado. Uso Java profissionalmente



Pontifícia Universidade Católica do Rio de Janeiro

4 – Você já teve algum treinamento formal em criptografia, segurança de redes ou segurança da informação? ___

Caso afirmativo, por favor, elabore sua resposta.

5 – Em sua organização, ou na última organização que você trabalhou, quem é/era responsável por inspecionar o código a fim de detectar vulnerabilidades de segurança?

( ) Ninguém

( ) O próprio desenvolvedor

( ) O líder do projeto ou gerente de desenvolvimento

( ) Um especialista de segurança da empresa

( ) Um especialista de segurança externo

6 – Na sua opinião, quem deveria ser responsável por verificar e detectar violações de alguma regra de criptografia?

( ) O próprio desenvolvedor

( ) O líder do projeto ou gerente de desenvolvimento

( ) Um especialista de segurança da empresa ou externo

( ) Um plugin da IDE ou uma Ferramenta Externa de detecção de códigos inseguros

( ) A própria classe que cifra ou decifra informações deve tratar desta verificação

Annex 3 Controlled Experiment Tasks

TASK 1

Please produce a code that encrypts and decrypts five times in a row the following plain text message: "Negotiations failed. War is imminent. Destroy all documents, passwords and keys". In each loop, print the encrypted message as a HEXADECIMAL string. Please print also the decrypted message to see if it matches the original message. The console output must be saved in a text file. Please do note the starting time of the task and the ending time. You are free to choose any encryption algorithm available in the Cipher Class (DES, 3DES, AES, RC4, Blowfish, Twofish). You may specify your own or generate the **encryption key**. But the **encryption key** MUST be the same for every loop.

Hints: import javax.xml.bind.DatatypeConverter

You can use DatatypeConverter.printHexBinary([] byte) to encode an array of bytes as a HEXADECIMAL string.

You are free to research on any web site or forum. You may also copy and paste code to complete the tasks. Do not turn the page until you complete this task.

TASK 2

Please produce a code that encrypts and decrypts five times in a row the following plain text message: “Negotiations failed. War is imminent. Destroy all documents, passwords and keys”. In each loop, print the encrypted message as a HEXADECIMAL string. Please print also the decrypted message to see if it matches the original message. The console output must be saved in a text file. Please note the starting time of the task and the ending time. You must use AES/CTR encryption algorithm. If you chose AES/CTR for the first task, use AES/CBC for this task. You may specify your own or generate the **encryption key**. But the **encryption key** MUST be the same for every loop.

Hints: import javax.xml.bind.DatatypeConverter

You can use DatatypeConverter.printHexBinary([] byte) to encode an array of bytes as a HEXADECIMAL string.

You are free to research on any web site or forum. You may also copy and paste code to complete the tasks. Do not turn the page until you complete this task.

TASK 3

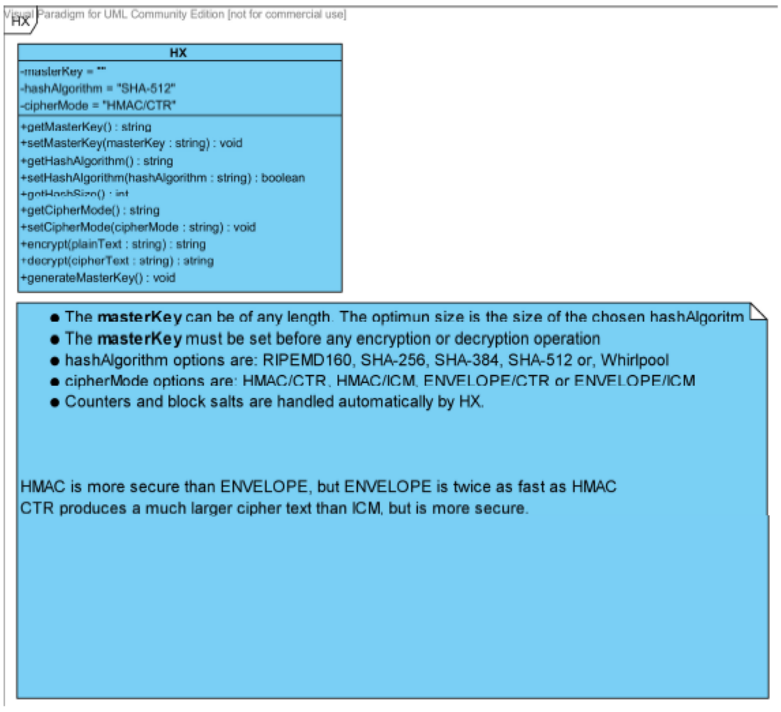
Please produce a code that encrypts and decrypts five times in a row the following plain text message: "Negotiations failed. War is imminent. Destroy all documents, passwords and keys". In each loop, print the encrypted message as a HEXADECIMAL string. Please print also the decrypted message to see if it matches the original message. The console output must be saved in a text file. Please note the starting time of the task and the ending time. You may specify your own or generate the **encryption key**. But the **encryption key** MUST be the same for every loop. You may either accept the defaults values of the HX Class or you may change them if you want.

Hints:

You must import puc.galgos.crypto.HX to use the HX class.

There is no example or forum to seek help for HX. Use the UML documentation of the class and the JAVADOC in order to complete the task.

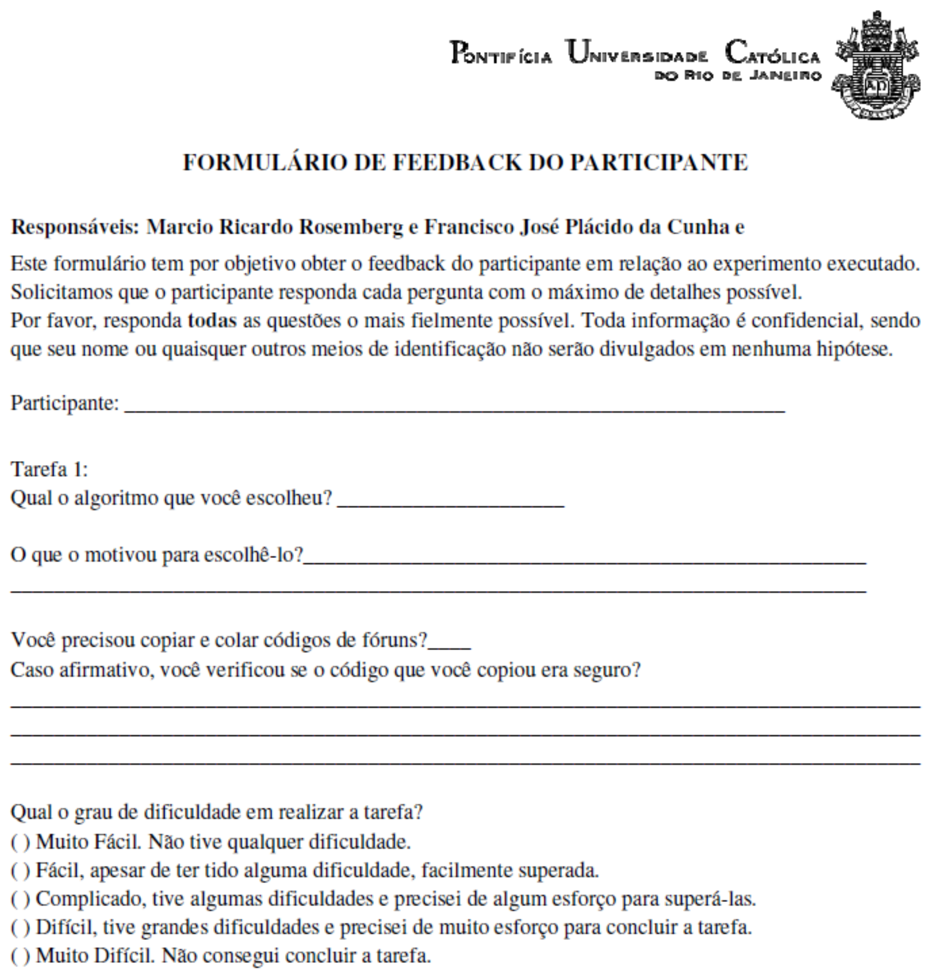
Do not forget to set the masterKey before invoking the encryption or decryption methods.


Visual Paradigm for UML Community Edition [not for commercial use]
HX
HX
-masterKey = ""
-hashAlgorithm = "SHA-512"
-cipherMode = "HMAC/CTR"
+getMasterKey() : string
+setMasterKey(masterKey : string) : void
+getHashAlgorithm() : string
+setHashAlgorithm(hashAlgorithm : string) : boolean
+getHashSize() : int
+getCipherMode() : string
+setCipherMode(cipherMode : string) : void
+encrypt(plainText : string) : string
+decrypt(cipherText : string) : string
+generateMasterKey() : void

- The **masterKey** can be of any length. The optimun size is the size of the chosen hashAlgoritm
- The **masterKey** must be set before any encryption or decryption operation
- hashAlgorithm options are: RIPEMD160, SHA-256, SHA-384, SHA-512 or, Whirlpool
- cipherMode options are: HMAC/CTR, HMAC/ICM, ENVELOPE/CTR or ENVELOPE/ICM
- Counters and block salts are handled automatically by HX.

HMAC is more secure than ENVELOPE, but ENVELOPE is twice as fast as HMAC
CTR produces a much larger cipher text than ICM, but is more secure.

Annex 4 Controlled Experiment Feedback Form



PONTIFÍCIA UNIVERSIDADE CATÓLICA DO RIO DE JANEIRO

FORMULÁRIO DE FEEDBACK DO PARTICIPANTE

Responsáveis: Marcio Ricardo Rosemberg e Francisco José Plácido da Cunha e

Este formulário tem por objetivo obter o feedback do participante em relação ao experimento executado. Solicitamos que o participante responda cada pergunta com o máximo de detalhes possível.
Por favor, responda **todas** as questões o mais fielmente possível. Toda informação é confidencial, sendo que seu nome ou quaisquer outros meios de identificação não serão divulgados em nenhuma hipótese.

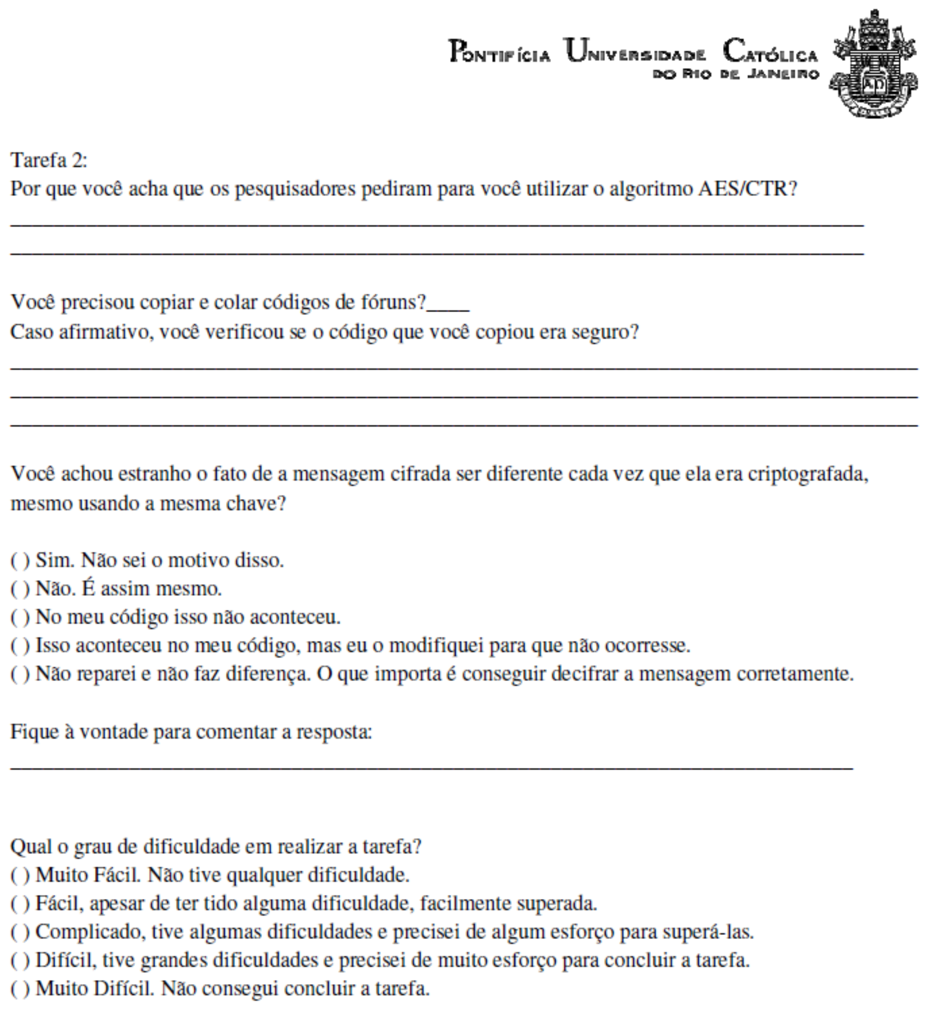
Participante: ____

Tarefa 1:
Qual o algoritmo que você escolheu? ______________

O que o motivou para escolhê-lo?____________________________________
____

Você precisou copiar e colar códigos de fóruns?____
Caso afirmativo, você verificou se o código que você copiou era seguro?
____
____
____

Qual o grau de dificuldade em realizar a tarefa?
( ) Muito Fácil. Não tive qualquer dificuldade.
( ) Fácil, apesar de ter tido alguma dificuldade, facilmente superada.
( ) Complicado, tive algumas dificuldades e precisei de algum esforço para superá-las.
( ) Difícil, tive grandes dificuldades e precisei de muito esforço para concluir a tarefa.
( ) Muito Difícil. Não consegui concluir a tarefa.



Pontifícia Universidade Católica do Rio de Janeiro

Tarefa 2:
Por que você acha que os pesquisadores pediram para você utilizar o algoritmo AES/CTR?
____
____

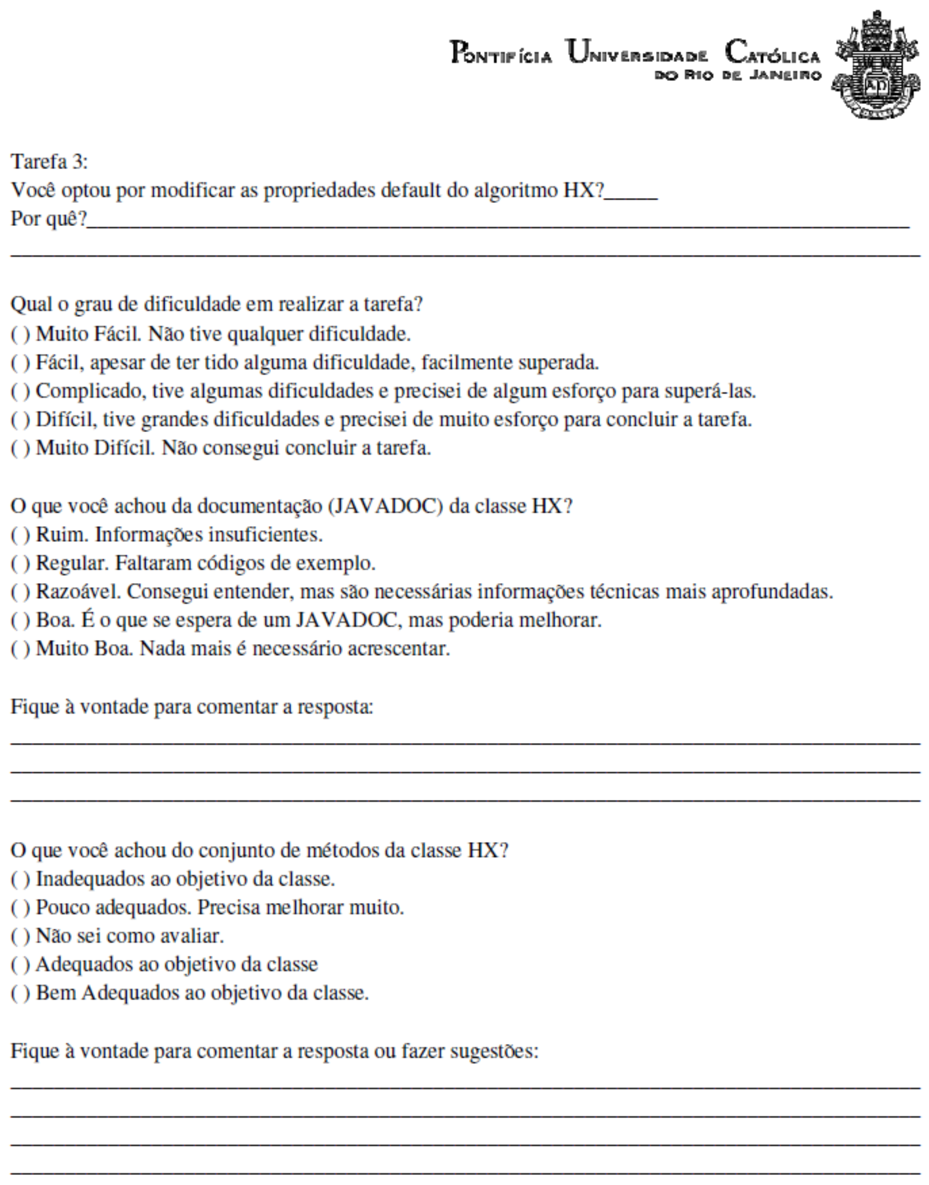
Você precisou copiar e colar códigos de fóruns?____
Caso afirmativo, você verificou se o código que você copiou era seguro?
____
____
____

Você achou estranho o fato de a mensagem cifrada ser diferente cada vez que ela era criptografada, mesmo usando a mesma chave?

( ) Sim. Não sei o motivo disso.
( ) Não. É assim mesmo.
( ) No meu código isso não aconteceu.
( ) Isso aconteceu no meu código, mas eu o modifiquei para que não ocorresse.
( ) Não reparei e não faz diferença. O que importa é conseguir decifrar a mensagem corretamente.

Fique à vontade para comentar a resposta:
____

Qual o grau de dificuldade em realizar a tarefa?
( ) Muito Fácil. Não tive qualquer dificuldade.
( ) Fácil, apesar de ter tido alguma dificuldade, facilmente superada.
( ) Complicado, tive algumas dificuldades e precisei de algum esforço para superá-las.
( ) Difícil, tive grandes dificuldades e precisei de muito esforço para concluir a tarefa.
( ) Muito Difícil. Não consegui concluir a tarefa.



PONTIFÍCIA UNIVERSIDADE CATÓLICA
DO RIO DE JANEIRO

Tarefa 3:
Você optou por modificar as propriedades default do algoritmo HX?_____
Por quê?____
____

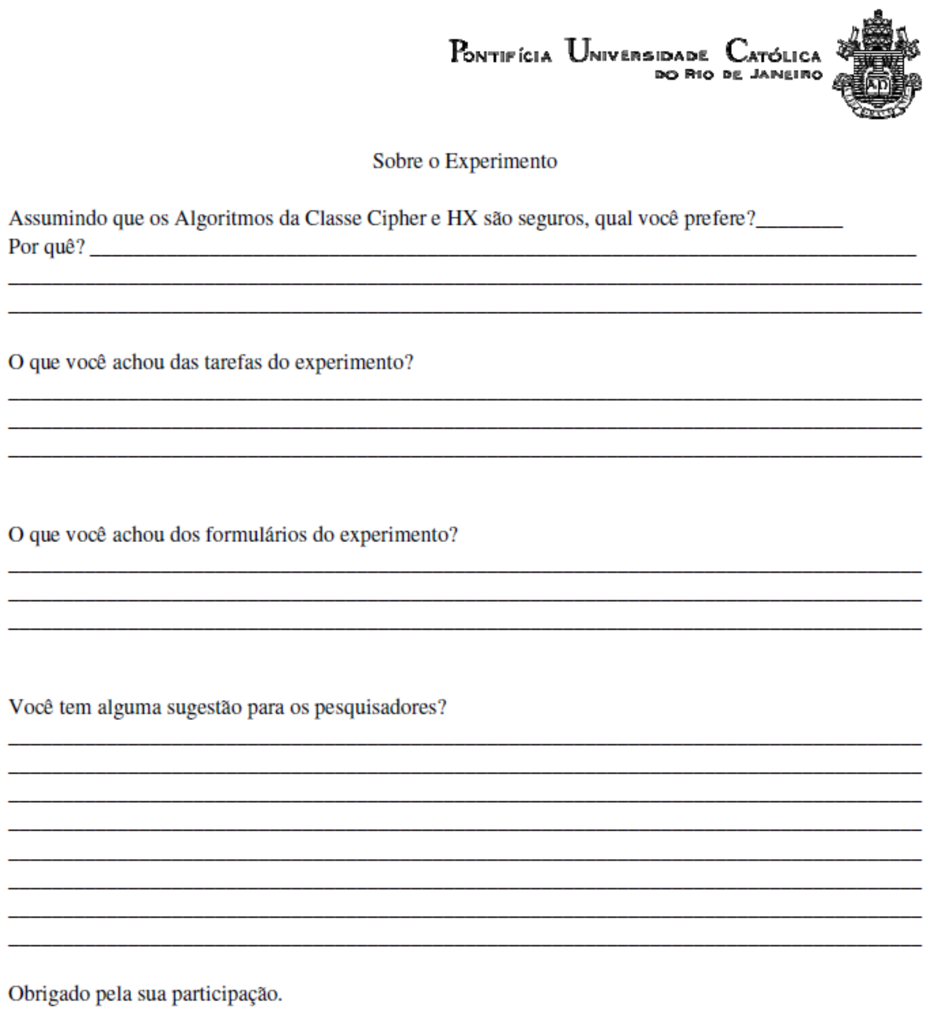
Qual o grau de dificuldade em realizar a tarefa?
( ) Muito Fácil. Não tive qualquer dificuldade.
( ) Fácil, apesar de ter tido alguma dificuldade, facilmente superada.
( ) Complicado, tive algumas dificuldades e precisei de algum esforço para superá-las.
( ) Difícil, tive grandes dificuldades e precisei de muito esforço para concluir a tarefa.
( ) Muito Difícil. Não consegui concluir a tarefa.

O que você achou da documentação (JAVADOC) da classe HX?
( ) Ruim. Informações insuficientes.
( ) Regular. Faltaram códigos de exemplo.
( ) Razoável. Consegui entender, mas são necessárias informações técnicas mais aprofundadas.
( ) Boa. É o que se espera de um JAVADOC, mas poderia melhorar.
( ) Muito Boa. Nada mais é necessário acrescentar.

Fique à vontade para comentar a resposta:
____
____
____

O que você achou do conjunto de métodos da classe HX?
( ) Inadequados ao objetivo da classe.
( ) Pouco adequados. Precisa melhorar muito.
( ) Não sei como avaliar.
( ) Adequados ao objetivo da classe
( ) Bem Adequados ao objetivo da classe.

Fique à vontade para comentar a resposta ou fazer sugestões:
____
____
____
____



Pontifícia Universidade Católica do Rio de Janeiro

Sobre o Experimento

Assumindo que os Algoritmos da Classe Cipher e HX são seguros, qual você prefere?________
Por quê? ____
____
____

O que você achou das tarefas do experimento?
____
____
____

O que você achou dos formulários do experimento?
____
____
____

Você tem alguma sugestão para os pesquisadores?
____
____
____
____
____
____
____
____

Obrigado pela sua participação.

Feedback Form Answers

Task 1

Participant's Number	**Submission Date**	**Which Algorithm did you choose?**	**What motivated you to choose that algorithm?**
4	2018-08-31 14:53:33	AES/CBC	AES/CBC is the most practical unless the project context requires another algorithm.
3	2018-08-31 14:39:51	AES/ECB Default	never used the Cipher Class before. It was the simplest to implement.
1	2018-08-31 14:31:43	AES/CBC	Easiest algorithm to find documentation and examples
5	2018-08-31 14:11:35	AES/CBC	the first algorithm found during a query in Google
2	2018-08-31 14:00:15	AES/CBC	Apear to be the most simple to implement. Had used AES before.
6	2018-08-20 20:23:53	AES/CBC	I quickly read about them and saw that important institutions used it
9	31/08/2018 15:20	AES/CBC	first algorithm found in a query
7	31/08/2018 15:13	DES	No specific reason
12	31/08/2018 15:09	AES/ECB Default	First found in a query
8	31/08/2018 15:04	AES/CBC	First algorithm found in a query
11	31/08/2018 15:00	AES/ECB Default	First algorithm found in a search engine query
10	20/08/2018 19:41	AES	Eu já ouvi falar nele, não conhecia os outros.

Participant's Number	Submission Date	Did you copied and pasted any code from forums, documentations or examples?	If you copied code from any source, did you check the code for any vulnerability ?	How do you classify the difficulty level of the task?
4	2018-08-31 14:53:33	Yes	Yes	Easy, even though I had some difficulty, easily overcame
3	2018-08-31 14:39:51	Yes	Yes	Complicated. I experienced some difficulty and I needed some effort to overcome
1	2018-08-31 14:31:43	Yes	No	Complicated. I experienced some difficulty and I needed some effort to overcome
5	2018-08-31 14:11:35	Yes	No	Easy, even though I had some difficulty, easily overcame
2	2018-08-31 14:00:15	Yes	No	Complicated. I experienced some difficulty and I needed some effort to overcome
6	2018-08-20 20:23:53	Yes	Yes	Easy, even though I had some difficulty, easily

				overcame
9	31/08/2018 15:20	Yes	Yes	Very easy. I did not have any difficulty
7	31/08/2018 15:13	Yes	No	Easy, even though I had some difficulty, easily overcame
12	31/08/2018 15:09	Yes	No	Easy, even though I had some difficulty, easily overcame
8	31/08/2018 15:04	Yes	No	Too difficult. I was unable to complete the task
11	31/08/2018 15:00	Yes	No	Easy, even though I had some difficulty, easily overcame
10	20/08/2018 19:41	Yes	No	Complicated. I experienced some difficulty and I needed some effort to overcome

Task 2

Participant's Number	Submission Date	Why do you think the researchers asked you to use AES-CTR algorithm?	Did you copied and pasted any code from forums, documentations or examples?
4	2018-08-31 14:53:33	Do not know	Yes
3	2018-08-31 14:39:51	To examine the volume of code necessary to perform the task	Yes
1	2018-08-31 14:31:43	Because of the difficulty level to use AES-CTR, which does not give a good result. It's not safe.	Yes
5	2018-08-31 14:11:35	Comparison with AES/CBC	Yes
2	2018-08-31 14:00:15	Comparison with the previous task	No
6	2018-08-20 20:23:53	I have no idea.	Yes
9	31/08/2018 15:20	Because it is more difficult to be broken.	Yes
7	31/08/2018 15:13	Not idea	Yes
12	31/08/2018 15:09	Do not know	Yes
8	31/08/2018 15:04	did not complete the task	No
11	31/08/2018 15:00	Because it is more complicated than the default AES	Yes
10	20/08/2018 19:41	Eu não sei, mas acredito que possa ser porque o AES puro sempre gera o mesmo "Hash" enquanto o AER/CTR gera "Hashes" diferentes para uma mesma entrada. E desta forma tendo o mesmo comportamento do HX.	Yes

Task 3

Participant's Number	Submission Date	Did you opt to modify the properties of the HX algorithm?	Why?
4	2018-08-31 14:53:33	No	wanted to examine the results with the default values.
3	2018-08-31 14:39:51	No	The default values were well suited to the task.
1	2018-08-31 14:31:43	No	Decided to trust the default values
5	2018-08-31 14:11:35	No	only changed the masterKey, which is necessary.
2	2018-08-31 14:00:15	No	Consulting JAVADOC, found the default values adequated
6	2018-08-20 20:23:53	No	If possible, I do not know how to do it.
9	31/08/2018 15:20	No	unfamiliarity with both the Class and with cryptographic algorithms
7	31/08/2018 15:13	No	did not specify
12	31/08/2018 15:09	No	did not specify
8	31/08/2018 15:04	No	Convenience
11	31/08/2018 15:00	No	Did not specify
10	20/08/2018 19:41	No	Ao meu ver as propriedades padrão já eram as melhores opções. Gostei muito dos Hints do Javadocs, ajudou bastante.

Participant's Number	Submission Date	How do you classify the difficulty level of the task? 3	What did you think of the JAVADOC documentation of the class HX?
4	2018-08-31 14:53:33	Very easy. I did not have any difficulty	Very good. It does not need any other information
3	2018-08-31 14:39:51	Very easy. I did not have any difficulty	Good. It's what is expected from a JAVADOC
1	2018-08-31 14:31:43	Easy, even though I had some difficulty, easily overcame	Acceptable. I was able to understand, but it lacks further technical information
5	2018-08-31 14:11:35	Very easy. I did not have any difficulty	Very good. It does not need any other information
2	2018-08-31 14:00:15	Very easy. I did not have any difficulty	Good. It's what is expected from a JAVADOC
6	2018-08-20 20:23:53	Easy, even though I had some difficulty, easily overcame	Good. It's what is expected from a JAVADOC
9	31/08/2018 15:20	Easy, even though I had some difficulty, easily overcame	Acceptable. I was able to understand, but it lacks further technical information
7	31/08/2018 15:13	Easy, even though I had some difficulty, easily overcame	Good. It's what is expected from a JAVADOC
12	31/08/2018 15:09	Easy, even though I had some difficulty, easily overcame	Good. It's what is expected from a JAVADOC
8	31/08/2018 15:04	Very easy. I did not have any difficulty	Good. It's what is expected from a JAVADOC
11	31/08/2018 15:00	Very easy. I did not have any difficulty	Very good. It does not need any other information
10	20/08/2018 19:41	Easy, even though I had some difficulty, easily overcame	Good. It's what is expected from a JAVADOC

Participant's Number	Submission Date	Please feel free to comment your answer	What did you think of the set of methods of the HX class?
4	2018-08-31 14:53:33	Found the methods self-explanatory enough. The JAVADOC was not needed.	Well suited to the goals of the class
3	2018-08-31 14:39:51	Found some methods, without the JAVADOC	Well suited to the goals of the class
1	2018-08-31 14:31:43	Took some time to understand what the objective of the hash algorithm. Found some grammatical errors. For some reason taught the setHashAlgorithm() method would also set the masterKey.	Suited to the goals of the class
5	2018-08-31 14:11:35	Liked the JAVADOC.	Undecided. I can't evaluate.
2	2018-08-31 14:00:15		Suited to the goals of the class
6	2018-08-20 20:23:53	I did not use the JAVADOC.	Undecided. I can't evaluate.
9	31/08/2018 15:20	Lack code examples	Suited to the goals of the class
7	31/08/2018 15:13		Suited to the goals of the class
12	31/08/2018 15:09		Suited to the goals of the class
8	31/08/2018 15:04	Needed examples. The documentation did not specify if the methods were static or not.	Well suited to the goals of the class
11	31/08/2018 15:00	Not difficulties whatsoever to use the Class	Well suited to the goals of the class
10	20/08/2018 19:41		Well suited to the goals of the class

Final Questions

Participant's Number	Submission Date	Assuming the algorithms of the Cipher class and the HX Class are safe, which do you prefer?	Why?
4	2018-08-31 14:53:33	Cipher	The source code of the Cipher class is not available. In information security, it is important to disclose sensitive algorithms for public knowledge.
3	2018-08-31 14:39:51	HX	Simplest to use and the cleanest the code produced.
1	2018-08-31 14:31:43	HX	The easiest to use
5	2018-08-31 14:11:35	HX	Found HMAC/CTR the strongest cipher mode
2	2018-08-31 14:00:15	Cipher	Slower to execute
6	2018-08-20 20:23:53	I do not know	I do not have the knowledge need to evaluate both of them.
9	31/08/2018 15:20	HX	Easier to use than the others
7	31/08/2018 15:13	HX	Simplest to use
12	31/08/2018 15:09	HX	Because HX is encapsulated.
8	31/08/2018 15:04	HX	Simplicity in the use
11	31/08/2018 15:00	HX	Much more easy to use
10	20/08/2018 19:41	HX	Achei muito mais fácil de utilizar. Pouco código e produz um resultado satisfatório.

Participant's Number	Submission Date	What was your impression of the tasks?	What was your impression of experiment's forms?
4	2018-08-31 14:53:33	liked the opportunity to remember some cryptography concepts, such as padding.	Well suited for the experiment
3	2018-08-31 14:39:51	Positive, since he had few knowledge of cryptography.	Well suited to the experiment proposal.
1	2018-08-31 14:31:43	Cool and Easy. Even though unexperienced with the Cipher Class, it was easy to find examples.	Meticulous. The researchers could have asked which Cryptography API the developers were familiar with.
5	2018-08-31 14:11:35	precise and direct questions, simple enough to accomplish the experiment goal.	Too complex and to detailed, but good.
2	2018-08-31 14:00:15	Simple tasks that required only the basic knowledge of cryptography	Simple an direct questions. On-line forms would be appreciated.
6	2018-08-20 20:23:53	It seems simple, but I did not get the objective of them.	They are clear and simple
9	31/08/2018 15:20		
7	31/08/2018 15:13		
12	31/08/2018 15:09		
8	31/08/2018 15:04		
11	31/08/2018 15:00		
10	20/08/2018 19:41		

Participant's Number	Submission Date	Do you have any suggestions for the researchers?
4	2018-08-31 14:53:33	
3	2018-08-31 14:39:51	
1	2018-08-31 14:31:43	Record the screen during the experiment.
5	2018-08-31 14:11:35	check his code before the evaluation of the feedback form answers.
2	2018-08-31 14:00:15	On-line forms, feedback forms after each task. To record the execution of the tasks
6	2018-08-20 20:23:53	I wish to, but unfortunately, I have not the experience or knowledge
9	31/08/2018 15:20	
7	31/08/2018 15:13	
12	31/08/2018 15:09	
8	31/08/2018 15:04	
11	31/08/2018 15:00	
10	20/08/2018 19:41	