# ON AUTHENTICATION OF LONG MESSAGES IN SHORT TIME FOR RESOURCE CONSTRAINED SENSOR NETWORKS

A Thesis
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Rucha Subodh Sule

In Partial Fulfillment
for the Degree of
MASTER OF SCIENCE

Major Department:
Electrical and Computer Engineering

April 2013

Fargo, North Dakota

# North Dakota State University
## Graduate School

**Title**

ON AUTHENTICATION OF LONG MESSAGES IN SHORT TIME FOR RESOURCE CONSTRAINED SENSOR NETWORKS

**By**

Rucha Subodh Sule

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

**MASTER OF SCIENCE**

SUPERVISORY COMMITTEE:

Dr. Rajendra Katti
Chair

Dr. Sudarshan Srinivasan

Dr. Changhui Yan

Approved:

April 4, 2013
Date

Dr. Rajendra Katti
Department Chair

# ABSTRACT

This thesis is a collection of three different research contributions targeted towards providing faster message authentication for long messages which have been recently accepted or submitted for publication. The first research work explores properties of Multiple Input Shift Register (MISR) as a universal hash function. We implemented a fixed length message authentication code (MAC) based on MISR in software. Signing or verification time of new MAC is two order less compared to existing MAC. The second contribution is a variable length MAC based on MISR for use in smart grid networks. We prove security of the MAC scheme and analyze its performance pertaining to smart grid application.

The third contribution suggests use of one-time signatures (OTS) from sigma protocols for multicast authentication in smart grid. The proposed scheme yields three order improvements in time performance at a very modest increase in signature size compared to currently best known OTS scheme.

# ACKNOWLEDGMENTS

I sincerely thank Professor Rajendra Katti, my advisor, who has provided continuous guidance throughout my master's program. I could not have been at this point today without his honest and generous support. I am also thankful to Dr. Kavasseri for introducing me to smart grid technology and providing guidance as and when required. I am grateful to my supervisory committee members, Dr. Srinivasan, and Dr. Yan for their guidance and advice.

I would like to thank my family and friends, who have always been greatly supportive. Thanks also to the faculty, staff, and students of the Electrical and Computer Engineering Department of NDSU for providing such an excellent friendly environment to learn and progress.

# TABLE OF CONTENTS

# LIST OF TABLES

# LIST OF FIGURES

# CHAPTER 1. INTRODUCTION

We live in the era of smart digital technology. Everything we use in day to day life can be controlled by our smart phone. We track train or bus using our phone, we find parking slot using our phone, we receive temperature and traffic information at single click using smart phone. In addition to this machine to machine (M2M) technology has made it possible to monitor home appliances sitting in the office. We are moving towards using smart meters that work on demand response mechanism and save energy. All this is made possible by use of sensor networks. Spatially distributed sensors monitor and record data and pass it along the network. This data needs to be secured before it is transmitted otherwise it becomes vulnerable to numerous kinds of attacks. Data security can be divided into mainly two areas, privacy and authenticity. In this work we will concentrate mainly on achieving authenticity. There are many well established cryptographic schemes that achieve authenticity. But, these schemes require lot of resources in terms of computation and storage. However, sensors have very limited resources and hence cannot use these regular methods. Hence, there have been surge of interest in finding lightweight authentication schemes for sensor networks. We will analyze existing schemes and propose new schemes that are more efficient in terms of computation time as well as storage and implementation.

In this chapter first a brief general background on lightweight authentication schemes is given. We also briefly introduce some of most significant existing methods. This chapter does not intend to provide in depth explanation about sensor authentication schemes as each chapter of the thesis consists of a stand-alone self-contained paper in which the required background and definitions are explained. We will also discuss the motivations of our research work and

highlight the contributions in a separate section of this chapter. At the end the organization of the thesis is briefly described.

## 1.1. Background

Sensor networks cover wide range of applications from Environment/Earth Monitoring, Industrial Monitoring, Military Monitoring, Agriculture, Tracking Systems, and Smart Home Monitoring. The sensor nodes have to work in resource constrained environment. They have very limited computational power, memory, on chip space, bandwidth and power source. They are not only resource constrained but most of them have heavy data like images. Hence they need to be treated differently. As sensor networks collect data that could be secret or sensitive, there is need to ensure integrity of this data. Not only in military applications but also in other applications malicious users for their own benefit might try to tamper the data sent between different nodes. Data integrity can be achieved using various authentication methods. Use of asymmetric key methods like RSA or El Gamal Signature schemes in sensor networks is out of question as they require lot of computational resources. However, symmetric key methods like HMAC sound like a feasible option but it also requires considerable amount of hardware and computation time. Therefore, researchers are moving towards a symmetric key approach of hashing the input message first into a small size message and then applying a cryptographic primitive to the hashed data.

The aim of a message authentication code is to prevent an adversary from modifying a message sent by one party to another, without the parties detecting that a modification has been made [1-1]. Any symmetric key authentication technique is commonly known as Message Authentication Code (MAC). Two users who wish to communicate in an authenticated manner begin by generating and sharing a secret key $k$ in advance of their communication. When one

party wants to send a message $m$ to the other, she computes a MAC tag (or simply a tag) $t$ based on the message and the shared key, and sends the message $m$ along with the tag $t$ to the other party. The tag is computed using a tag-generation algorithm that will be denoted by Mac; rephrasing what we have already said, the sender of a message $m$ computes $t \leftarrow Mac_k(m)$ and transmits $(m, t)$ to the receiver. Upon receiving $(m, t)$, the second party verifies whether $t$ is a valid tag on the message $m$ (with respect to the shared key) or not. This is done by running a verification algorithm $Vrfy$ that takes as input the shared key as well as a message $m$ and a tag $t$, and indicates whether the given tag is valid [1-1].

As mentioned earlier hash and encrypt approach is the best known way of providing authentication for long messages in resource constrained environment. The hash functions used in this context are called universal hash function. Term universal basically means that these hash functions distribute their input evenly among their output. In this research we propose a new universal hashing technique and explore its use in smart grid networks.

We will also propose a new one time signature scheme for multicast authentication is smart grid networks. This scheme on the contrary makes use of public key signatures in the innovative way. In this scheme we make use of sigma protocol to construct a public key signature. This approach does not require as heavy computations as traditional public key signatures. A public key signature scheme is used in the following way. One party $S$, who acts as the sender, runs $Gen(1^n)$ to obtain keys $(pk, sk)$ . The public key $pk$ is then publicized as belonging to $S$. e.g., $S$ can put the public key on its webpage or place it in some public directory. We assume that any other party is able to obtain a legitimate copy of $S$'s public key. When $S$ wants to transmit a message $m$, it computes the signature $\sigma \leftarrow Sign_{sk}(m)$ and sends $(m, \sigma)$. Upon receipt of $(m, \sigma)$, a receiver who knows $pk$ can verify the authenticity of $m$ by checking

whether $Vrfy_{pk}(m, \sigma) = 1$. This establishes both that $S$ sent $m$, and also that $m$ was not modified in transit [1-1]. We construct one time signature using sigma protocol and prove that even though it is a public key method it will prove to be very good choice for multicast authentication in smart grid.

## 1.2. History

Universal hash functions and their use in MAC were first suggested by Carter and Wegman in 1981[1-2]. They for the first time introduced an idea of applying cryptographic primitive to output of universal hash in order to compute MAC. Universal hash functions are widely preferred in sensor networks due to their ability to compute MAC in efficient time while providing unconditional security. Cost of computing tag using universal hash functions is sum of cost of computing hash and cost of applying cryptographic primitive to output of hash. However, if hash compresses message well then the second step does not consume much of time [1-3]. Hence quest of finding efficient MAC is reduced to finding efficient universal hashing technique. Since then there had been surge of interest in finding more and more efficient universal hash constructions. These constructions are aimed to reduce signing and verification delays, hardware resources, tag size and power consumption. The simplest construction amongst these constructions is Cryptographic-CRC proposed in [1-4]. They propose a construction where Linear Feedback Shift Register (LFSR) with slight modification is used as $\epsilon$-balanced universal hash function. However LFSR takes as many cycles as length of message to hash the message. In [1-5] use of MISR in MAC is suggested for first time but they did not see that MISR are in fact universal hash functions and can be used for message authentication in numerous ways. Use of MISR introduces parallelism in Cryptographic-CRC scheme and hence improves the time performance. The construction is very simple and requires very low hardware resources.

Smart grid sensor networks are gaining popularity in recent years. They have separate standards (IEEE P2030 SG) for maintaining security. Figure 1.1 describes overall architecture of smart grid networks. In [1-6] new utility computer network security management and authentication in smart grid operations is proposed. However, they do not consider communication between all types of smart grid network nodes. In [1-7], number of digital security issues that need to be addressed for SG communication are discussed. They point out vulnerabilities in combining SCADA/EMS systems with existing information networks. Metke Et al. [1-8] pointed out need of message authentication code in smart grid networks. They mention that use of existing schemes won't be a smart choice for authentication in smart gird networks instead there is need for a scheme that is faster and can adapt according to requirements. In [1-9] a new framework for message authentication between different nodes of smart grid network based on Diffie-Hellman key exchange and HMAC-RIPEMD is proposed. However, RIPEMD is not considered as secure hash function. We suggest replacing HMAC in their scheme with our variable length MAC and achieve two order better performances in terms of signing and verification delay.

Smart Gird networks consist of different types of messages. The type of message can be classified into uni-cast or multicast depending on type of application. In multicast communication single transmitter sends a message intended for multiple receivers. These messages many times may contain sensitive control/command messages. This necessitates need of multicast message authentication in smart grid networks. Multicast authentication in smart grid networks is achieved through one time signature (OTS). An OTS scheme [1-10, 1-11] makes use of cryptographically secure one way functions so that single key can be used to sign

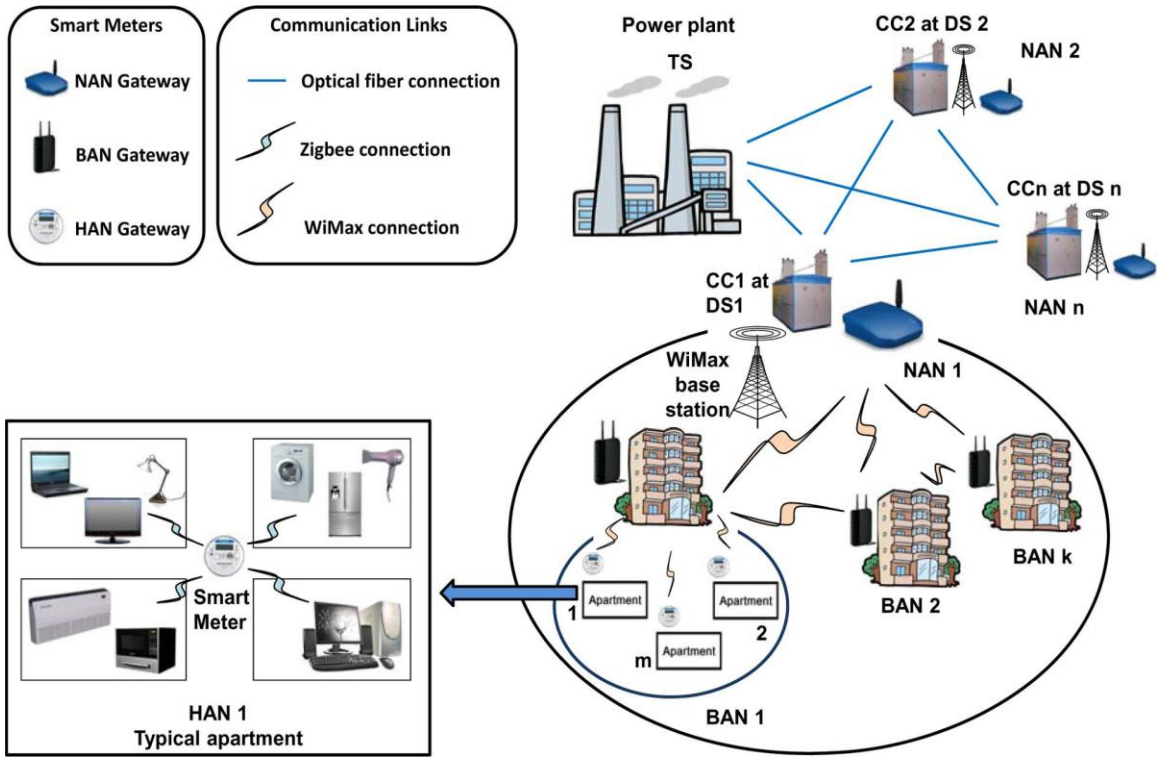


Figure 1.1. Smart Grid Communication Framework ([1-9] Copyright © 2011, IEEE).

multiple messages. These schemes are fast but have large signature size and hence cannot be used in smart gird communication. The Bins and Balls (BiBa) [1-12] scheme reduces signature size at the expense of increased signing delay. HORS (Hash to Obtain Random Subset) [1-13] overcomes the deficiencies in BiBa. However, HORS requires large key sizes. This limits its use in field devices (sensors). To address drawbacks of HORS the Tunable Signing and Verification scheme (TSV) is proposed in [1-14]. The scheme can be adapted based on application, more computations are performed either at the sender or receiver. Though this scheme is efficient for communication between sensors and base station, it won't be a good choice for communication between two sensor nodes. Hence we suggest entirely new approach of using sigma protocols for computing OTS. Sigma protocols are widely used for e-banking, e-voting, e-credentials but we seek their use in smart grid networks. We achieve three orders improvement in terms of signing

and verification delays as compared to TSV. Also, pre-computation and storage cost is significantly reduced.

## 1.3. Motivations and Contributions

The thesis is a collection of three different papers targeted towards providing fast authentication for long messages in short time which have been recently accepted or submitted for publication as three different papers. The first paper proposes that Multiple Input Shift Register (MISR) is $\epsilon$-balanced universal hash function. We perform small modification in the existing $\epsilon$-balanced universal hash function, Cryptographic-CRC [1-4] to construct our universal hash function MISRH. In [1-4] hash function $h_p(M)$ for any message $M$ of binary length $d$ is defined as $M(x)x^n \, mod \, p(x)$, where $p(x)$ is irreducible polynomial of degree $n$ over $GF(2)$. We split the message $M$ of binary length $d$ to hash into $L$ data streams each $s$-bit long, $M = M_1 \parallel M_2 \parallel \cdots \parallel M_i \parallel \cdots \parallel M_L$, $|M_i| = s, L \leq n, M_{L+1} \parallel \cdots \parallel M_n$ are all zero message streams. The binary sequence $M_i$ can be represented using a polynomial $M_i(x)$. The MISRH is computed using $\left(\sum_{i=1}^{n} M_i(x)x^{n-i}\right) mod \, p(x)$, where $p(x)$ is irreducible polynomial of degree $n$ over $GF(2)$.

Cryptographic-CRC requires $d$ clock cycles to compute hash on $d$-bit long message. We will reduce this time by the factor of $L$. The fastest known universal hash function is UMAC, it requires 0.52 cycles/byte for achieving security of the order of $2^{-60}$. Whereas, MISRH requires mere 0.13 cycles/byte to achieve the same security. In this paper we prove that MISRH is $\epsilon$-almost universal for $\epsilon = 1/2^{n-1}$, and $\epsilon$-balanced for $\epsilon = s/2^{n-1}$. We also propose different ways of constructing MAC from MISRH and prove that construction $(r, f(r) \oplus MISRH(M))$, where $r$ is counter and $f \in F$ (Pseudo Random Function PRF) is $\epsilon$-opt secure MAC. To implement this construction 25648 logic elements are required. Hence, MAC scheme based on

MISRH will be a very good choice for message authentication in resource constrained environment.

In second paper we propose a variable length MAC using MISRH for use in smart grid networks. The best known message authentication infrastructure was proposed by Fauda et. Al in [1-9]. They use Diffie-Hellman key exchange to establish a shared secret between home area network node and building area network node. They use this shared secret to encrypt and sign the message. They suggest using HMAC-RIPEMD128 for signing the message. However, RIPEMD128 is not considered as secure hash. We propose replacing HMAC with our new variable length MAC, $F_k(MISRH(M||pad))$. We analyze the security of this new MAC scheme and prove that it is existentially unforgeable under chosen message attack. We compare performance of this scheme with HMAC and show that our scheme performs two orders better in terms of signing and verification delay. Also, this scheme provides a fixed length tag hence reduces the communication overhead by significant amount too.

In third paper we propose a multicast authentication scheme for use in smart grid networks using sigma protocols. In smart grid networks, multicast authentication is achieved using one time signature. Numerous methods have been proposed to reduce the cost of computation, cost of key generation and distribution, key length, signature length and storage cost. Some methods are good in terms of computation time but they require lengthy keys or produce lengthy signatures. Other schemes which provide short signatures require more computation time. The best known OTS scheme today is TSV. It adapts based on type of network configuration it is used in. It requires higher computations on either sender or receiver node. Therefore we propose a new approach of using sigma protocol based one-time signature in this scenario. We construct an OTS using witness hiding sigma protocol. Its security is same as

probability of solving discrete log problem. We provide background on sigma protocol and prove security of OTS using sigma protocol. We compare our scheme with TSV and show that our scheme provides three order reductions in signing and verification cost. It also provides two order reductions in pre-computation and storage cost. However, our scheme increases signature size by four fold compared to TSV. Even so OTS based on sigma protocol seems to be a very good option for multicast authentication in smart grids.

In first paper provided in this thesis I worked with my advisor Dr Katti. I worked on proving that MISR are universal hash functions. I implemented the MAC based on MISRH in software and evaluated its performance. I also did literature survey on universal hash functions and compared hardware performance of MISRH with existing universal hash functions. We have submitted this paper to 16$^{th}$ EUROMICRO Conference on Digital System Design (DSD). In second paper of this thesis I worked with my advisor Dr Katti to construct a variable length MAC and prove its security. I worked on evaluating the performance of the scheme and comparing it with HMAC. The literature survey related to smart grid networks is done by Dr Kavasseri. The paper appears in 2012 IEEE power and engineering society general meeting. In third paper I worked mostly on evaluating the performance of the scheme and doing part of literature survey. We submitted this paper to IEEE Transactions on Smart Grid.

## 1.4. Thesis Organization

This thesis contains three different research works targeted towards providing fast authentication for long messages in short time which have been recently accepted or submitted for publication as three different papers. Chapter 2 of this thesis consists of first paper under title "Fast Message Authentication for Long Messages in Resource Constrained Environment Using New Universal Hash: MHMAC". We have submitted this paper for publication to 16$^{th}$

EUROMICRO Conference on Digital System Design (DSD). Chapter 3 of this thesis consists of second paper titled “A Variable Length Fast Message Authentication Code for Secure Communication in Smart Grids”. The paper has been accepted in 2012 IEEE power and engineering society general meeting. Chapter 4 of this thesis covers third paper under title “Multicast Authentication in the Smart Grid with One-Time Signatures from Sigma-Protocols”. The paper was submitted to IEEE Transactions on Smart Grid. Finally in chapter 5 we draw general conclusions about our research.

# CHAPTER 2. FAST MESSAGE AUTHENTICATION FOR LONG MESSAGES IN RESOURCE CONSTRAINED ENVIRONMENT USING UNIVERSAL HASH: MHMAC

We have submitted this paper for publication to 16th EUROMICRO Conference on Digital System Design (DSD). The authors of the paper are Rajendra S Katti, Rucha S. Sule.

## 2.1. Abstract

In this paper we consider fast authentication of long messages. We prove that Multiple Input Shift Register (MISR) are $\epsilon$-balanced hash function and hence can be used in a Message Authentication Code (MAC). The message $M$ to hash is split into $L \leq n$ data streams each $s$-bit long, $M = M_1 \parallel M_2 \parallel \cdots \parallel M_i \parallel \cdots \parallel M_L$, $|M_i| = s$, $M_{L+1} \parallel \cdots \parallel M_n$ being all zero message streams. These $M_i$s are given as input to MISR, which performs kind of division by polynomial of degree $n$ over $GF(2)$ to give $n$-bit hash. A cryptographic primitive is applied to the output of MISR to get MAC. We implemented this scheme in software to evaluate the performance of the scheme. Results of implementation show that it is possible to authenticate a message of size 4MB in 8mSec. We also insist that as the construction is very simple, it will prove to be hardware efficient in terms of both implementation and performance. The scheme will find its use in sensor networks and multimedia networks.

## 2.2. Introduction

Message Authentication Codes (MACs) are used to authenticate messages in the symmetric key setting. A tag is sent along with a message to a receiver who verifies that the message was not tampered using the tag. In many communication networks the communicated messages are lengthy which in turn necessitates the existence of fast MACs. Moreover, in many

networks, such as sensor networks, messages are sent very frequently. This fact stresses the need for MAC that takes multiple messages at the same time and generates a single tag for all the messages in efficient time.

In this paper, we consider the problem of authentication of long messages with small tag in very less number of computations. When we say less computations we are mainly referring to hardware computations, but our scheme will prove to be a good option in case of software implementation too. We also make sure that the scheme is easy to understand as well as implement. We will prove that MISR is in fact a universal hash function and evaluate its performance by implementing the scheme in software. The MAC based on MISR requires 8mSec to authenticate a message of size 4MB. There are numerous ways in which message authentication code can be constructed from any universal hash function. We will list these methods and show, how a simple $\epsilon$-*opt secure* message authentication code MHMAC can be constructed from our hash MISRH.

We achieve the speed by introducing parallelism where, a long data file can be split into $L$ shorter blocks of length $s$ and a single tag $t$ can be computed for the $L$ blocks. In this case the time needed to compute the tag and the verification time are reduced by a factor of $L$. To achieve the security of the order of $2^{-60}$ (probability of collision), our hash function requires 0.13 cycles/byte (apx. 128*8/60 $\approx$ 17 cycles for message of length 128Bytes). For fixed message length, number of cycles per byte decreases as security requirement increases, which also results in increase in the hardware. All this is realized by simple modification in Cryptographic-CRC method [2-4]. Cryptographic-CRC is a well-known universal hashing technique but it requires 6cycles/byte [2-10] to compress the message. We reduce it by factor of $L$ while maintaining same hardware complexity. Most of the existing methods use very complicated ways to construct

universal hash functions. However, use of MISR as universal hash function not only improves the hashing time but also provides a simpler way of constructing universal hash function. In following subsection we will review the universal hashing approach and how it can be used in a secure MAC setting.

### 2.2.1. Universal Hashing Approach

We will first start with definition of hash functions. In [2-3] hash functions are defined as follows:

*Definition 2.2.1.1:*

A hash family is a four-tuple $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$, where the following conditions are satisfied:

- $\mathcal{X}$ is a set of possible messages
- $\mathcal{Y}$ is a finite set of possible messages
- $\mathcal{K}$, the keyspace, is a finite set of possible keys

For each $k \in \mathcal{K}$, there is a hash function $h_k \epsilon \mathcal{H}$. Each $h_k : \mathcal{X} \rightarrow \mathcal{Y}$.

In above definition $\mathcal{X}$ could be a finite or infinite set; $\mathcal{Y}$ is always a finite set. If $\mathcal{X}$ is a finite set, then the corresponding hash function is called a compression function. In this situation it is assumed that $\mathcal{X} \geq \mathcal{Y}$. A pair $(x, y) \in \mathcal{X} \times \mathcal{Y}$ is said to be valid under key $k$ if $h_k(x) = y$. Let $\mathcal{F}^{\mathcal{X},\mathcal{Y}}$ denote the set of all functions from $\mathcal{X}$ to $\mathcal{Y}$. Suppose that $|\mathcal{X}| = N$ and $|\mathcal{Y}| = M$. Then it is clear that $\left|\mathcal{F}^{\mathcal{X},\mathcal{Y}}\right| = M^N$. Any such hash family $\mathcal{F} \subseteq \mathcal{F}^{\mathcal{X},\mathcal{Y}}$ is termed an (N,M)-hash family.

When a hash function is used for cryptographic purposes, it's security is decided based on following criteria. If hash function is considered to be secure, it should be the case that following three problems are difficult to solve.

- Preimage:

  Instance: A hash function $h: \mathcal{X} \rightarrow \mathcal{Y}$ and an element $y \in \mathcal{Y}$.

  Find: $x \in \mathcal{X}$ such that $h(x) = y$.

- Second Preimage:

  Instance: A hash function $h: \mathcal{X} \rightarrow \mathcal{Y}$ and an element $x \in \mathcal{X}$.

  Find: $x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x') = h(x)$.

- Collision:

  Instance: A hash function $h: \mathcal{X} \rightarrow \mathcal{Y}$.

  Find: $x, x' \in \mathcal{X}$ such that $x' \neq x$ and $h(x') = h(x)$.

A hash function for which Collision cannot be efficiently solved is often said to be *collision resistant*.

Concept of universal hash functions is introduced by Carter and Wegman [2-5] in 1979. They state that any class of function that is *universal*$_2$ has the property that given any sample, a randomly chosen member of that class will be expected to distribute the sample evenly. He defines a parameter $\delta_f(x, x')$ such that, given a function $f$, and $x, x' \in \mathcal{X}$. $\delta_f(x, x') = 1$ if $x' \neq x$ and $f(x') = f(x)$, otherwise $\delta_f(x, x') = 0$. If *f*, $x$ or $x'$ is replaced in $\delta_f(x, x')$ by set of elements, then sum of all elements in respective sets is computed. Thus, if $H$ is a collection of hash functions, $x \in \mathcal{X}$ and $S \subset \mathcal{X}$ then $\delta_H(x, S)$ means $\sum_{f \in H} \sum_{x' \in S} \delta_f(x, x')$. The quantity $\delta_H(x, x')$ counts number of functions in $H$ for which $x$ and $x'$ collide. If $H$ is class of functions

from $\mathcal{X} \to \mathcal{Y}$ then $H$ is *universal*$_2$ if for all $x, x'$ in $\mathcal{X}$, $\delta_H(x, x') \leq \frac{|H|}{|\mathcal{Y}|}$. Clearly for any function to be *universal*$_2$ number of collisions should be less than one $|\mathcal{Y}|^{\text{th}}$ of the number of functions in the hash family. They also introduce few examples of such functions in this work. We will prove in section 2.4.2 that our hash function will have collision with probability approximately$1/2^n \ \forall\, n \gg 1$, where $n$ is the length of a tag.

Since [2-5] defined *universal*$_2$ hash functions, different authors proposed modified versions of *universal*$_2$ hash functions as follows:

*Definition 2.2.1.2:* [2-3]

Suppose that $(\mathcal{X}, \mathcal{Y}, \mathcal{K}, \mathcal{H})$ is an *(N,M)* hash family. This hash family is *strongly universal* provided that the following condition is satisfied for every $x, x' \in \mathcal{X}$ such that $x' \neq x$, and for every $y, y' \in \mathcal{Y}$*:*

$$|\{K \in \mathcal{K}: h_K(x) = y, h_K(x') = y'\}| = \frac{|\mathcal{K}|}{M^2}$$

To prove any class of hash function to be *strongly universal* it is sufficient to show that there exists a unique function that maps $x, x' \in \mathcal{X}$, where $x' \neq x$ *to* $y, y' \in \mathcal{Y}$.

*Definition 2.2.1.3:* [2-7]

A $\epsilon$*-almost universal*$_2$ hash function, $h: \mathcal{K} \times \mathcal{X} \to \mathcal{Y}$, must satisfy that

$\forall\, x, x' \in \mathcal{X}, (x \neq x'): \delta_H(x, x') \leq \epsilon |H|$.

*Definition 2.2.1.4:* [2-7]

A $\epsilon$-*almost strongly universal*$_2$ hash function, $h: \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$, must satisfy that

$\forall\ x \in \mathcal{X} \setminus \{0\}$ and $y \in \mathcal{Y}$: $|\{h \in H: h(x) = y\}| \leq |H|/|\mathcal{Y}|$.

$\forall\ x, x' \in \mathcal{X}\ (x \neq x')$ and $y, y' \in \mathcal{Y}$, $|\{h \in H: h(x) = y, h(x') = y'\}| \leq \epsilon|H|/|\mathcal{Y}|$

If $K$ is length of key $k$, such that $\mathcal{K} = \{0,1\}^K$,$\mathcal{X} = \{0,1\}^N$, $\mathcal{Y} = \{0,1\}^M$ then

*Definition 2.2.1.5:* [2-4], [ 2-8]

A $\epsilon$-*balanced* universal hash function, $h: \mathcal{K} \times \mathcal{X} \rightarrow \mathcal{Y}$, must satisfy that

$\forall\ x \in \mathcal{X} \setminus \{0\}$ and $y \in \mathcal{Y}$: $Pr_{\{k \in \mathcal{K}\}}[h(k, x) = y] \leq \epsilon$.

In section 2.4.2 we will analyze our hash function MISRH according to these definitions and show that it is in fact a *collision resistant*, *universal*$_2$, $\epsilon$-*almost universal* and $\epsilon$-*balanced* hash function.

### 2.2.2. Universal Hashing and Authentication

Over the past three decades MACs based on Universal Hash Family [2-5] are preferred in sensor networks because of their ability to provide fast computation with unconditional security. In [2-6] Carter and Wegman further introduced an authentication scheme based on *universal*$_2$ hash functions. Any authentication scheme based purely on universal hash functions cannot be used for authentication of multiple messages. In order to enable hash function based schemes to authenticate multiple messages, [2-6] suggest to XOR, the output of hash function with random sequence as in one time pad system. In this case, probability of forgery is same as that of selection of random pad for MAC. We follow similar approach to prove security of our MAC.

As pointed out by [2-9], the speed of universal hash based MAC depends mainly on the speed of the hashing step and the speed of encryption step. But if the hash function compresses message well then the encryption should not take long, simply because it is a short string that is being encrypted. Hence quest of finding fast MAC based on the universal hash function reduces to finding fast universal hash itself. Hence, many authors proposed different designs for fast universal hashing. Here, we will provide an overview of the methods from which our hash is adapted. All other methods are described in detail in section 2.3.

Krawczyk [2-4] first suggested the use of Linear Feedback Shift Registers (LFSR) in constructing MAC. This method is well known as "Cryptographic CRC", which has very fast hardware implementation and reasonably fast software implementations; it needs 6cycles/byte, as shown by Shoup in [2-10]. In [2-4] hash function $h_p(M)$ for any message $M$ of binary length $m$ is defined as $M(x)x^n \, mod \, p(x)$, where $p(x)$ is irreducible polynomial of degree $n$ over $GF(2)$. This paper also presents a Toeplitz-Matrix based hashing using LFSR which is also considered to be very hardware efficient. In [2-11], [2-12], [2-13], [2-14] hash schemes were proposed that performs a division by a random irreducible polynomial. Rogaway bucket hashing in [2-15] was the first universal hash family targeted for fast software implementation. It hashes in about 1.5-2.5 cycles/byte [2-15]. Paper [2-16] presented a bucket hashing algorithm with smaller key size. Halevi and Krawczyk in [2-17] present fast method for implementing Modular Multiplication based universal hash which utilizes properties of MMX architecture achieving speed of about 1.5-3 cycles/byte. Further [2-9] gives even faster method utilizing SIMD architecture properties; it was the first paper that described complete construction of MAC while analyzing the efficiency of software implementation. They introduce parallelism by dividing the message into $w$-bit words and then compute the hash function NH. They achieve speed of 0.52

cycle/byte [2-9] for achieving security of $2^{-60}$. They were the first to apply pseudorandom function to the output of hash function to compute MAC. In [2-18] Palash Sarkar gives new multilinear hash based on reapplication of LFSR function. However this method requires $l$ shift and multiply operations for $l$-bit message. It requires approximately 128*128/2048 = 8 cycles/byte to hash 2kB message when q=2 and n=128. The collision probability is $(1/2^n)$, where $n$ is length of the key.

As mentioned earlier we will achieve better hardware performance. To compute a hash for 2kB message with n=128, we will require mere 128 cycles resulting in performance of 0.0625 cycles/byte. To achieve this we will perform a small modification in the basic "Cryptographic CRC" construction to introduce parallelism to form our universal family of hash functions MISRH and apply cryptographic primitive at the output of hash to generate MHMAC. The message $M$ to hash is split into $L$ data streams each $s$-bit long, $M = M_1 \parallel M_2 \parallel \cdots \parallel M_i \parallel \cdots \parallel M_L$, $|M_i| = s, L \leq n, M_{L+1} \parallel \cdots \parallel M_n$ are all zero message streams for $L < n$. The binary sequence $M_i$ can be represented using a polynomial $M_i(x)$. The MISRH is computed using formula, $\left(\sum_{i=1}^{n} M_i(x)x^{n-i}\right) mod\ p(x)$, where $p(x)$ is irreducible polynomial of degree $n$ over $GF(2)$. And finally MAC can be computed using $F_k\left(\left(\sum_{i=1}^{n} M_i(x)x^{n-i}\right) mod\ p(x)\right)$ or $F_k(r) \oplus \left(\sum_{i=1}^{n} M_i(x)x^{n-i}\right) mod\ p(x)$. As pointed out by UMAC, the security can further be improved by appending random nonce to message before giving it as input to pseudorandom function. There are many other ways of constructing MAC from universal hash. We will discuss these ways in detail in section 2.4.3. Main goal of our method is to perform fast authentication of large messages while keeping the construction simple. We also succeed in maintaining the scheme provably secure.

Rest of the paper is organized as follows: In section 2.3 number of existing universal hash functions are described. Here, we will also list other universal hash based MACs as well as few other MACs. In section 2.4 we will describe our new hash function MISRH and analyze in which category it fits into. We will also describe simple MAC construction MHMAC based on MISRH and prove its security. Section 2.5 will analyze software and hardware performance of MHMAC and compare it with existing schemes. Finally Section 2.6 will conclude this paper.

## 2.3. Related Work

Concept of *universal$_2$* Hash was introduced by Carter and Wegman in [2-5]. In [2-6] they further introduced an authentication scheme based on *universal$_2$* hash functions. Any authentication scheme based purely on universal hash functions cannot be used for authentication of multiple messages. In order to enable hash function based schemes to authenticate multiple messages, [2-6] suggest to XOR, the output of hash function with random sequence as in one time pad system. However with their scheme secret key grows out of proportion if large number of messages has to be authenticated [2-19]. To avoid this, Brassard suggests another way of generating the pseudorandom sequence to be XORed. Since then most of the work concentrated on finding efficient ways of generating pseudorandom sequences for such applications. In 1994 Stinson [2-7] gave few more classes of universal hash functions, he called them *$\epsilon$-almost universal$_2$* and *$\epsilon$-almost strongly universal$_2$*. He further proposed MAC based on these hash functions.

On the other hand Krawczyk [2-4] suggests an innovative way of using LFSR for hashing and then XORing the output with pseudorandom pad. He also present a Toeplitz-Matrix based hashing using LFSR techniques which takes $2n$ bits to represent hash functions compared to $n * m$ bits in original multiplicative hash(Here $n$ is length of output of hash function and $m$ is

length of message). In his following paper [2-8] he presents more generalized approach for constructing such Toeplitz Matrix. In [2-11], [2-12] hash schemes were proposed that performed a division by a random irreducible polynomial. In [2-10] Shoup evaluates different methods of implementing provably secure MAC using universal hash, and set of pseudorandom functions as well as permutations. He presents various algorithms for efficient implementation of these methods. These families had shorter outputs and were therefore possibly more practical.

Universal hash based MAC methods replaced traditional ones whenever there was need of fast processing in either software or hardware. Halevi and Krawczyk in [2-17] present fast method for implementing Modular Multiplication based Universal Hash utilizing properties of MMX architecture. Further [2-9] gives even faster method utilizing SIMD architecture properties, it was the first paper that described complete construction of MAC while analyzing the efficiency of software implementation. They suggest decomposing hash into small segments, then apply NH to each segment separately, and further apply HMAC to concatenation of these hashed values and nonce to compute tag. In [2-20] Black gave a formal proof for the security of a MAC construction in which a pseudorandom function is applied to the output of a universal hash function. Here, NH [2-9] is computed by further fragmenting the message segment and applying corresponding part of key to it. Each function is named by a string $K$ of $nw$-bits, where $n$ is block size ($n \geq 2$) and $w$ is wordsize ($w \geq 1$). Probability of Collision is $2^{-w}$, which can be improved by using Toeplitz construction. In 2000, [2-21] gave a scheme with better collision probability than MMH or NH. In [2-22], Kaps et. al. introduce WH, a power optimized version of NH, which is efficient for hardware implementation in ultra-low power devices. They maintain the probability of collision to be $2^{-w}$. In [2-18] Palash Sarkar gives a new universal

hash family. It requires $l$ shift and multiply operations to compress $l$-bit message with collision probability of $(1/q^n)$, where $n$ is length of key and $q$ is base of field $IF_q^n$.

NMAC and HMAC constructions which are based on "collision-resistant hash functions" [2-23] were introduced by Bellare et al. in 1996 [2-24]. Both NMAC and HMAC constructions can be used for variable-length messages. Recently, there has been a surge of interest in aggregate MACs [2-25], [2-26]. The goal in this line of research is to reduce the number of tags routed in a network in which many nodes send messages to a single destination node and communication is an expensive resource. The proposed solution is to combine the tags of multiple messages together, such that the resulting tag is verifiable by the destination party. Katz and Lindell were the first to propose a formal proof for the security of aggregate MACs [2-25]. In these methods, a short tag may be produced but the tag generation and verification times are proportional to the number of messages. [2-43]

We combine the hardware efficiency of LFSR schemes with the novel approach of universal hash based MACs to present a new MAC scheme that computes MAC tag for long messages in very short time. For this purpose an MISR should behave like a secure hash function. MISRs are widely used as compactors where similar phenomenon like collision occurs and is widely known as aliasing. Many researchers have worked on finding the aliasing probability for MISRs.

In the late-90s, [2-27] suggested a novel approach of using coding theory to find out the aliasing probability of an MISR. They utilized properties of maximum distance separable (MDS) codes to compute the aliasing probability. After that [2-28], [2-29], [2-32], [2-33] extended this work to provide a closed form expression for the aliasing probability of an MISR either by using

coding theory or wireless channel models. In [2-34] Morii and Iwasaki provided an extension for work in [2-28]. They claim that, Reed Solomon (RS) codes used in [2-28] are nothing but shortened RS codes resulting in MDS codes. A comparison of aliasing probability of multiple MISRs and a single MISR is done in [2-35]. Khan and Bushnell [2-31] provide results regarding asymptotic aliasing probability. They observed using different simulation results that the aliasing probability rapidly converges to a very low value.

Back in 1990 the approach of using a Markov model to find the aliasing probability was presented in [2-38]. They provide proper statistical theory that explains dependence of aliasing probability on main MISR features such as its length and feedback network. In the same period researchers at IBM made use of Markov chains to claim that the aliasing probability depends on the correlation of data at different inputs [2-30]. In 2005 Hadijicostis [2-39] made use of a Markov chain to calculate the exact aliasing probability for any test sequence. A similar analysis for LFSRs is performed in [2-40], whereas [2-41] examines aliasing in case of q-ary symmetric error model.

In 1993 Pilarski, Kameda and Ivanov [2-36] used MISR for sequential faults and presented an equation for aliasing probability related to it. Future work may require demonstrating an approach to select the secret polynomial $p(x)$ in our method. However we have not considered this problem in this paper. Work in [2-37] may prove useful in that regard. Hardware efficiency of MISRs is analyzed in [2-42] by Savir. He analyzed the effect of reducing the MISR size on the aliasing probability. These studies might be useful while considering hardware implementation of our scheme. But this is out of the scope of this paper. However, MISRs will definitely prove to be hardware efficient compared to regular cryptographic hash functions.

## 2.4. Our Contributions

### 2.4.1. New Hash Family: MISRH(M)

MISRH is first step for achieving fast and secure MAC. Here we describe in short the design of hash function and which category it fits into. This hash can be used in multiple ways to obtain a secure MAC.

As mentioned earlier this hash is obtained by introducing parallelism in Cryptographic-CRC described in [2-4]. The $d$ bit long message $M$ to hash is split into $L$ data streams each $s$-bit long, $M = M_1 \parallel M_2 \parallel \cdots \parallel M_i \parallel \cdots \parallel M_L$, $|\mathrm{M_i}| = \mathrm{s}, \mathrm{L} \leq \mathrm{n}$, $M_{L+1} \parallel \cdots \parallel M_n$ is all zero message stream for $L < n$. The binary sequence $M_i$ is converted to a polynomial $M_i(x)$ whose coefficients are equal to the binary sequence $M_i$. For example $1011$ is $x^3 + x + 1$ (each bit from the rightmost is multiplied by successive powers of $x$ and the sum of all these powers of $x$ is the polynomial). Hence each of $L$ data streams can be represented using $L$ polynomials each of degree s. Let $p(x)$ be any irreducible polynomial of degree $n \geq L$ over $GF(2)$. Now, $MISRH(M)$ can be computed as,

$$MISRH(M) = R(x) = \left( \sum_{i=1}^{n} M_i(x) x^{n-i} \right) mod\ p(x)$$

The coefficients of $R(x)$ form a $n$-bit binary sequence $R$, which is the output of our hash function. The irreducible polynomial $p(x)$ represents a hash function $MISRh_p$. The $MISRH$ is $(m, n)$ family of hash functions $MISRh_p$ where, $m = L * s = d$. The operation $\left(\sum_{i=1}^{n} M_i(x) x^{n-i}\right) mod\ p(x)$ can be easily implemented by the $n$-stage MISR with $M_i$ as input at

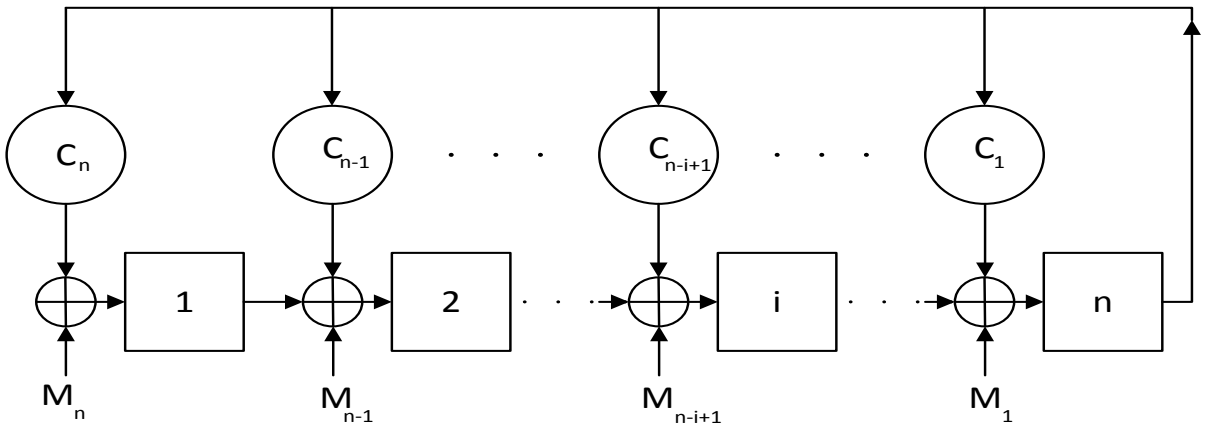


Figure 2.1. Multiple Input Shift Registers (MISR).

$i^{th}$ flip-flop as shown in Figure 2.1. The contents of flip-flops after $s$-bits of $M_i$s have entered MISR are equal to $R$ [2-2].

The novelty of the method is that only $s$ cycles of CPU operations are required to compress $(L * s)$-bit message unlike cryptographic CRC where, $(L * s)$ cycles are required. Hence our method performs $L$ times faster than cryptographic CRC. Each function $MISRh_p$ can be represented with $n$-bits. The polynomial $p(x)$ can be easily changed by changing values of $C_i$. Basic hardware implementation of $n$-bit $MISRH$ will require $13n$ gates. Which clearly indicates it will be a very good choice for hardware resource constrained applications.

### 2.4.2. Type of Hash Family MISRH

As mentioned in section 2.2.1 there are different types of Universal Hash Families, we need to identify in which type our hash family fits into. In this section we will analyze properties of MISR and prove it to be *Collision Resistant*, $\epsilon$*-almost universal*$_2$, and $\epsilon$*-balanced*. MISR can be proved *universal*$_2$ provided number of elements in input is multiple of number of elements at output.

### 2.4.2.1. Collision Resistant

As mentioned in section 2.3, MISR are traditionally used in hardware testing, where similar phenomenon as collision occurs. This phenomenon is known as aliasing and it is well known fact that probability of aliasing is approximately $1/2^n$.

### 2.4.2.2. Strongly Universal

In order to prove that a family of hash function MISRH is *strongly universal*, there is need to show that there exists a unique irreducible polynomial $p(x)$ of degree $n$, that divides two polynomials $M(x) = \sum_{i=1}^{n} M_i(x)x^{n-i}$ and $M'(x) = \sum_{i=1}^{n} M'_i(x)x^{n-i}$ of degree $(s + n - 1)$ for all $M(x) \neq M'(x)$ to give $R(x)$ and respectively $R'(x)$. Given $M(x)$, $M'(x)$, $R(x)$, and $R'(x)$ we need to find $p(x)$, such that $M(x) + R(x) = Q(x) * p(x)$ and $M'(x) + R'(x) = Q'(x) * p(x)$. Here, $Q(x)$ and $Q'(x)$ are quotients obtained in respective divisions. Also, note that because all operations are in $GF(2)$, minus $(-)$ sign can be replaced by plus $(+)$ sign. It is clear from the equations that there need not be only one polynomial of degree $n$ that satisfies this condition. For example, if degree of $M(x)$ and $M'(x)$ is $2n$, which when divided by two irreducible polynomials of degree $n$, give remainders $R(x)$ and resp. $R'(x)$, then either one of the irreducible polynomials will give same quotients and remainders for given $M(x)$ and $M'(x)$. Hence for sure MISRH is not *strongly universal*. However, it can be made *strongly universal* by restricting value of $s$ to be less than $n + 1$.

### 2.4.2.3. Universal$_2$

In order to prove that a family of hash function MISRH is *universal$_2$*, we need to show that the value of $\delta_H(x, x') \leq |H|/|\mathcal{Y}|$. Here, $\delta_H(x, x')$ represents number of functions in $H$ for which $x$ and $x'$ collide [2-43]. In our case $|H|$ is number of all irreducible polynomials of degree

$n$ over $GF(2)$, and $|\mathcal{Y}|$ is number of all polynomials of degree $n$ over $GF(2)$. Therefore according to [2-4], $|H| = 2^{n-1}/n$ , $|\mathcal{Y}| = 2^n$ and hence, $|H|/|\mathcal{Y}| = 1/2n$.

*Theorem 2.4.2.1:* [2-43]

For any class of hash functions from $\mathcal{X}$ to $\mathcal{Y}$ there exists distinct elements $x$ and $x'$ such that

$\delta_H(x, x') \geq \frac{(|\mathcal{X}|-|\mathcal{Y}|)}{|\mathcal{Y}|(|\mathcal{X}|-1)}|H|$. Equality holds if $|\mathcal{X}|$ is multiple of $|\mathcal{Y}|$.

In our case $|\mathcal{X}|$ is number of elements in input. Any message to the hash function MISRH could be at most $sn$-bits long. Hence, $|\mathcal{X}| = 2^{sn}$, $|\mathcal{Y}| = 2^n$, and $|H| = 2^{n-1}/n$. Therefore, $\frac{(|\mathcal{X}|-|\mathcal{Y}|)}{|\mathcal{Y}|(|\mathcal{X}|-1)} = \frac{2^{ns}-2^n}{2^n(2^{ns}-1)} = \frac{2^n(2^s-1)}{2^n(2^{ns}-1)} = \frac{(2^s-1)}{(2^{ns}-1)} \approx \frac{1}{2^n} \forall s \neq 0, s \gg 1, n \gg 1$. Therefore we can write, $\frac{(|\mathcal{X}|-|\mathcal{Y}|)}{|\mathcal{Y}|(|\mathcal{X}|-1)}|H| \approx \frac{1}{2n}$. Hence, $\delta_H(x, x') \geq \frac{1}{2n}$. This implies that a minimum value $\delta_H(x, x')$ can achieve will be $\frac{1}{2n} = |H|/|\mathcal{Y}|$. Therefore we can say that MISRH will act as *universal₂* hash functions when $|\mathcal{X}|$ is multiple of $|\mathcal{Y}|$.

### 2.4.2.4. $\epsilon$-almost Universal$_2$

To prove that MISRH is *$\epsilon$-almost universal₂*, we need to show that $\delta_H(x, x') \leq \epsilon|H|$. In our case, $1/2n < (1/2^{n-1})(2^{n-1}/n)$. Hence, MISRH is *$\epsilon$-almost universal₂*, with $\epsilon = 1/2^{n-1}$ when $|\mathcal{X}|$ is multiple of $|\mathcal{Y}|$. However, if that is not the case then MISRH can still be proved *$\epsilon$-almost universal₂* in following manner. From definition of $\delta_H$, we can write $\delta_H(M, M') = |\{h_p \in H: h_p(M) = h_p(M')\}|$. Note that $h_p(M) = h_p(M')$ implies that $\sum_{i=1}^{n}(M_i(x) - M_i'(x))x^{n-i} \equiv 0(mod\ p(x))$. Here, $\sum_{i=1}^{n}(M_i(x) - M_i'(x))x^{n-i}$ can have at most $(n + s - 2)/n$ factors of degree $n$. Therefore there are at most $(n + s - 2)/n \approx \frac{s}{n} \forall s \gg n$ polynomials, $p(x)$,

that divide the polynomial $\sum_{i=1}^{n}(M_i(x) - M_i'(x))x^{n-i}$. Note that this follows from the fact that degree of $\sum_{i=1}^{n}(M_i(x) - M_i'(x))x^{n-i}$ is $(n + s - 2)$ and degree of $p(x)$ is $n$. $\delta_H(M, M')$ is number of polynomials that make $h_p(M) = h_p(M')$. Thus, $\delta_H(M, M') \leq s/n$. As stated earlier, $|H| = (2^{n-1}/n)$. Therefore, $\epsilon = s/2^{n-1}$ and MISRH is *$\epsilon$-almost universal$_2$*.

### 2.4.2.5. $\epsilon$-almost Strongly Universal$_2$

To show that MISRH is *$\epsilon$-almost strongly universal$_2$*, we need to prove that

$\forall\ x \in \mathcal{X} \setminus \{0\}$ and $y \in \mathcal{Y}$: $|\{h \in H: h(x) = y\}| \leq |H|/|\mathcal{Y}|$.

$\forall\ x, x' \in \mathcal{X}\ (x \neq x')$ and $y, y' \in \mathcal{Y}$, $|\{h \in H: h(x) = y, h(x') = y'\}| \leq \epsilon|H|/|\mathcal{Y}|$

In our case ,$\forall\ x \in \mathcal{X} \setminus \{0\}$ and $y \in \mathcal{Y}$, $h(x) = y$ *iff* $M(x) mod\ p(x) = R(x)$ *iff* $p(x)$ divides $M(x) + R(x)$. Here, $M(x) = \sum_{i=1}^{n} M_i(x)x^{n-i}$ as mentioned earlier. *Let* $Q(x) = M(x) + R(x)$. Cleary $Q(x)$ is non-zero polynomial of degree at most $n + s - 2$, and $p(x)$ is a polynomial of degree $n$ that divides $Q(x)$. Because of unique factorization property, $Q(x)$ has $n + s - 2/n \approx s/n\ \forall s \gg n$ irreducible factors of degree $n$. Hence we can say, there are $s/n$ functions in $H$ that map $M(x)$ *to* $R(x)$. However there are $2^{n-1}/n$ irreducible polynomials of degree $n$, or $2^{n-1}/n$ elements in $H$. Therefore, $Pr\{h(x) = y\} = \frac{\frac{s}{n}}{2^{n-1}/n} = s/2^{n-1}$. Hence, $|\{h \in H: h(x) = y\}| = s/n \geq 1/2n = |H|/|\mathcal{Y}|$ because $s \geq 1$. This clearly implies that our function is not *$\epsilon$-almost strongly universal$_2$*.

### 2.4.2.6. $\epsilon$-balanced

From the analysis performed in above paragraph, it is clear that $\Pr\{h_p(M) = R\} \leq s/2^{n-1}$. Therefore, clearly MISRH is $\epsilon$-*balanced* with $\epsilon = s/2^{n-1}$. Hence achieving better security as compared to Cryptographic CRC in which case $\epsilon \approx ns/2^{n-1}$.

## 2.4.3. Message Authentication Code based on MISRH : MHMAC

As proved in above subsection, MISRH can be used as either *collision resistant* hash function or *universal*$_2$, $\epsilon$-*almost universal*$_2$ and $\epsilon$-*balanced* hash function. Given this, immediate question is how message authentication code can be constructed from MISRH. This can be done by applying cryptographic primitive to the output of hash function. In [2-10], different ways of constructing MAC from universal hash are discussed. Simplest way of constructing MAC will be computing MAC as $(r, f(r) \oplus h_p(M))$, where *r* is counter and $f \in F$ (Pseudo Random Function PRF). In the same work, they prove that DES can be used as PRF. It won't be difficult to prove the same for AES. However, in that case MAC can be $(r, f(r) \oplus h_1(r) \oplus h_p(M))$. Here, $H_1$ is $\epsilon$-AXU family of hash functions mapping $n$-bit input to $n$-bit output. In order to have stateless MAC it is possible to compute MAC as $(r, f(r\alpha + s\beta) \oplus e(r\beta + s) \oplus h_p(M))$, where $f, e \in F$, and $r, s \in GF(2^n)$. In all these cases, we can prove that our scheme is $\epsilon$-opt-secure as of [2-4]. Otherwise, we can opt for applying pseudorandom function (PRF) at the output of hash, and go for $PRF(Hash)$ or $PRF(Hash, Nonce)$ as in [2-9]. In either case MHMAC will prove to be fastest of all, because of use of MISR. In this paper we will explore the $(r, f(r) \oplus h_p(M))$ option and prove that this MAC is $\epsilon$-opt secure. Also, [2-44] uses $PRF(h_p(M))$ to construct a MAC, and proves it to be existentially unforgeable under chosen message attack. In this paper

we will consider only fixed length MAC but it is easy to construct a variable length MAC using MISRH and prove it secure as done in [2-45]. The MAC construction π is as follows:

### 2.4.3.1. Construction π

- *Gen:* On input $n$ (the security parameter), choose $k \leftarrow \{0,1\}^n$ uniformly at random and a secret irreducible polynomial $p(x)$ over $GF(2)$ of degree $n$. Note that this step is done only once and hence the key is $(k, g(x))$.
- *Mac:* On input $k \in \{0,1\}^n$, irreducible polynomial $p(x)$ over $GF(2)$ of degree *n*, a value of counter $r$, and a message *M* of length $d = s.L$, $(M \in \{0,1\}^d)$ where $L \leq n$, split *M* into *L* parts each of length *s*, such that $M = (M_1, \ldots, M_L)$. Set $(M_{L+1}, \ldots, M_n)$ to zero. Input all $M_i$'s into MISR with characteristic polynomial $p(x)$. Contents of the flip flops after *s* cycles is called $MISRH(M) = \left(\sum_{i=1}^{n} M_i(x)x^{n-i}\right) mod\ p(x) = R$, $(R \in \{0,1\}^n)$. Output the $tag\ t = (r, F_k(r) \oplus R)$.
- *Vrfy:* On input $k \in \{0,1\}^n$, counter $r$, irreducible polynomial $p(x)$ over $GF(2)$ of degree *n*, a message $M \in \{0,1\}^d$, and a tag $t \in \{0,1\}^n$, split *M* into *L* parts each of length *s*,$(M_1, \ldots, M_L)$, where $L \leq n$. Set $(M_{L+1}, \ldots, M_n)$ to zero. Input all $m_i'$s into MISR with characteristic polynomial $p(x)$. Output 1 if and only if $t := F_k(r) \oplus MISRH(M)$.

Remark:

Above MAC construction is for fixed length messages with $d=L*s$ bits. The above definition can also be changed to a variable length MAC but we do not consider this extension in this paper. Please refer to [2-45] for details on variable length MAC using MISR.

### 2.4.4. Security of MHMAC

Above construction can be proved *$\epsilon$-opt secure* similar to that of cryptographic CRC. To prove that we consider an experiment $Mac - forge_{\mathcal{A},\pi}(n)$ in which adversary $\mathcal{A}$ has knowledge of construction $\pi$ but he is oblivious to secret values $k$, and $p(x)$. He is given access to an oracle which on input of message $M$ of his choice outputs corresponding tag $t$. Given, $M, t = (r, F_k(r) \oplus MISRH(M))$ if adversary can successfully output a valid tag $t' = (r, F_k(r) \oplus MISRH(M'))$ for any message of his choice $M'$. Then adversary becomes successful in the experiment $Mac - forge_{\mathcal{A},\pi}(n)$. $Mac - forge_{\mathcal{A},\pi}(n) = 1$ iff, i) $Vrfy(M', t') = 1$ ii) $M \neq M'$.

*Definition 2.4.4.1:*

A message authentication code $\Pi = (Gen, Mac, Vrfy)$ is said to be *$\epsilon$-opt secure*, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, $Pr[Mac - forge_{\mathcal{A},\pi}(n) = 1] \leq \epsilon$.

*Definition 2.4.4.2:* [2-4]

A necessary and sufficient condition for a family $H$ of hash functions to be *$\epsilon$-opt secure* is

$$\forall\, M_1 \neq M_2, \qquad c, Pr_h\{h(M_1) \oplus h(M_2) = c\} \leq \epsilon$$

*Definition 2.4.4.3:* [2-4]

If $H$ is $\oplus$-*linear* then $H$ is *$\epsilon$-opt secure* if and only if $H$ is *$\epsilon$-balanced*.

*Theorem 2.4.4.1:* [2-4]

A family of functions $H$ is $\oplus$-*linear* if for all $M,M'$ we have $h(M \oplus M') = h(M) \oplus h(M')$.

*Proof:*

$$h(M) + h(M') = \left( \left( \sum_{i=1}^{n} M_i(x) x^{n-i} \right) + \left( \sum_{i=1}^{n} M_i'(x) x^{n-i} \right) \right) mod\ p(x)$$

$$h(M) + h(M') = \left( \sum_{i=1}^{n} M_i(x) x^{n-i} + M_i'(x) x^{n-i} \right) mod\ p(x)$$

$$h(M) + h(M') = \left( \sum_{i=1}^{n} (M_i(x) + M_i'(x)) x^{n-i} \right) mod\ p(x)$$

$$h(M) + h(M') = h(M \oplus M')$$

Because, our scheme is $\oplus$-*linear*, we need to prove that it is $\epsilon$-*balanced*. But, as proved in section 2.4.2.6 our scheme is $\epsilon$-*balanced*. Hence we can say our scheme $\epsilon$-*opt secure.*

By Definition 2.4.4.2, our scheme MISRH is $\epsilon$-*opt secure* for $\epsilon \approx s/2^{n-1}$. For fixed value of *n*, security will degrade as *s* increases. Hence for L= n = 128 we can authenticate 2KB message in 128 cycles providing a security of the level $2^{-121}$.

## 2.5. Performance of MHMAC

### 2.5.1. Software Performance

We implemented our scheme MHMAC on a win32 with Intel core 2 duo 2.1GHz machine using C++ and evaluated it's time performance. We used 128bit MISR with

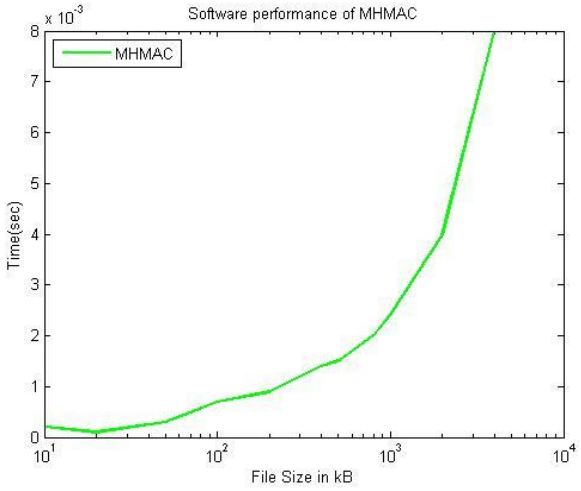

Figure 2.2. Software Performance of MHMAC.

characteristic polynomial $x^{128} + x^7 + x^2 + x$. As discussed earlier our scheme proves to be a very good choice for long messages. Figure 2.1 shows time performance of our scheme. We used crypto++ to implement 128 bit AES used as pseudorandom function. The scheme requires 8msec to compute a tag for message of size 4MB while achieving security of the order of $2^{-110}$. This clearly says that our scheme will be very good choice for fast authentication of long messages in short time while achieving high level of security.

### 2.5.2. Hardware Performance

Hardware implementation of MISR will require 1 cycle per $L$ bits of message to hash $sL$-bit message. For 100ns clock, 128 bit Rijndael AES takes about 320ns to encrypt a message [2-46]. Hence to compute MHMAC of $sL$-bit message we will require (s*100+320)ns. Figure 2.2 shows hardware performance of MHMAC for L=n=128. As shown in Figure 2.2 MHMAC will require 3.2ms to sign a message of 4MB. Thus as mentioned earlier our scheme will prove to be a very good choice for fast authentication of long messages. The configuration mentioned above will require (13*128) logic elements to implement MISRH and (20k+128+156+3700) logic elements [2-46] to implement AES. Hence total 25648 logic elements. As discussed in section

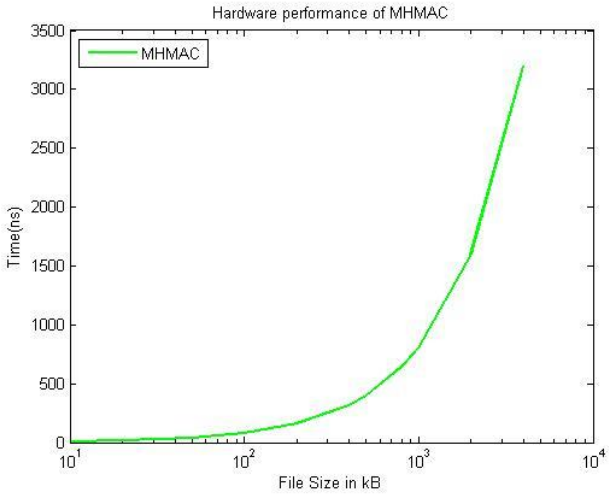

Figure 2.3. Hardware Performance of MHMAC.

2.2.2, our method requires mere 0.13 cycles for computation of hash function. When compared with existing methods, Figure 2.3 illustrates it very clearly that our method MISRH is faster than all existing methods. Table 2.1 shows this comparison in terms of cycles/bytes for n=128. Table 2.1 clearly illustrates that our hashing technique is faster than existing popular universal hashing techniques.

Table 2.1. Comparison with Existing Universal Hashing Techniques.

| Method | Cycles/Byte |
|---|---|
| MISRH | 0.0625 |
| Cryptographic CRC | 6 |
| Roagway Bucket Hashing | 1.5-2.5 |
| MMH | 1.5-3 |
| NH (UMAC) | 0.52 |
| Palash Sarkar (Multilinear Hash) for q=2 | 8 |

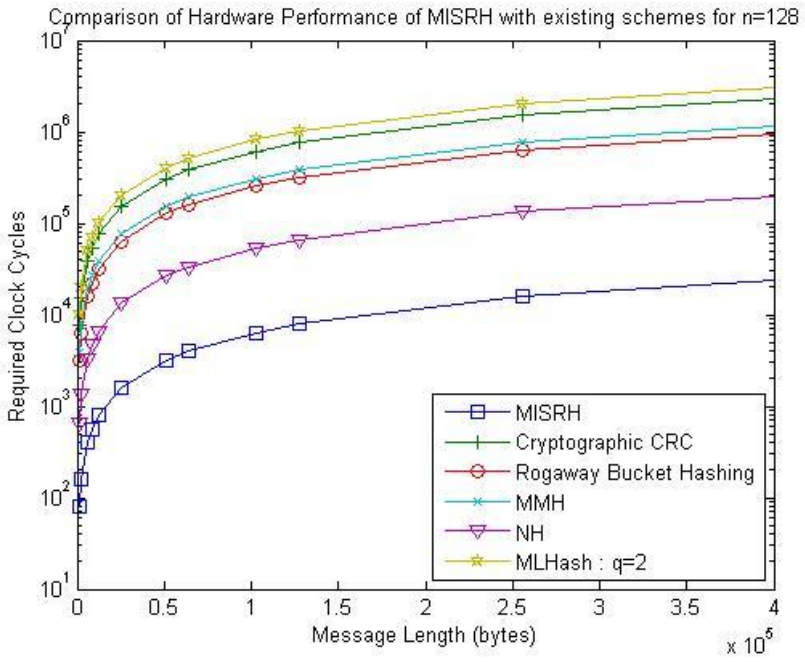


Figure 2.4. Comparison of Hardware Performance of MISRH with Existing Techniques, n=128.

## 2.6. Conclusion

Results from section 2.5 show that, the construction MISRH is not only faster but is simple and can be implemented in hardware with few logic elements. MISRH being *universal*$_2$, *$\epsilon$-almost universal*, and *$\epsilon$-balanced* hash function, it can be used in message authentication codes. The simplest construction of message authentication code can be proven $\epsilon$-opt secure. Therefore hash function MISRH and MAC based on MISRH will be a very good choice for fast authentication of long messages in resource constrained environment.

## 2.7. References

[2-1] Rao, Thammavarapu RN, and Eiji Fujiwara. "Error control coding for computer systems." *Prentice-Hall Inc.* (1989).

[2-2] Abramovici, Miron, Melvin A. Breuer, and Arthur D. Friedman. *Digital systems testing and testable design*. Vol. 2. New York: Computer science press, 1990.

[2-3] Stinson, Douglas Robert. *Cryptography: theory and practice*. CRC press, 2006.

[2-4] Krawczyk, Hugo. "LFSR-based hashing and authentication." In *Advances in Cryptology—CRYPTO'94*, pp. 129-139. Springer Berlin Heidelberg, 1994.

[2-5] Carter, J. Lawrence, and Mark N. Wegman. "Universal classes of hash functions." *Journal of computer and system sciences* 18, no. 2 (1979): 143-154.

[2-6] Wegman, Mark N., and J. Lawrence Carter. "New hash functions and their use in authentication and set equality." *Journal of computer and system sciences*22, no. 3 (1981): 265-279.

[2-7] Stinson, Douglas R. "Universal hashing and authentication codes." *Designs, Codes and Cryptography* 4, no. 3 (1994): 369-380.

[2-8] Krawczyk, Hugo. "New hash functions for message authentication." In *Advances in Cryptology—EUROCRYPT'95*, pp. 301-310. Springer Berlin Heidelberg, 1995.

[2-9] Black, John, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. "UMAC: Fast and secure message authentication." In *Advances in Cryptology—CRYPTO'99*, pp. 216-233. Springer Berlin Heidelberg, 1999.

[2-10] Shoup, Victor. "On fast and provably secure message authentication based on universal hashing." In *Advances in Cryptology—CRYPTO'96*, pp. 313-328. Springer Berlin Heidelberg, 1996.

[2-11] Bernstein, Daniel J. "The Poly1305-AES message-authentication code." In *Fast Software Encryption*, pp. 32-49. Springer Berlin Heidelberg, 2005.

[2-12] Taylor, Richard. "An Integrity Check Value Algorithm for Stream Ciphers." In*Proceedings of the 13th Annual International Cryptology Conference on Advances in Cryptology*, pp. 40-48. Springer-Verlag, 1993.

[2-13] Johansson, Thomas. "A shift register construction of unconditionally secure authentication codes." *Designs, Codes and Cryptography* 4, no. 1 (1994): 69-81.

[2-14] Kabatianskii, Gregory A., Ben Smeets, and Thomas Johansson. "On the cardinality of systematic authentication codes via error-correcting codes."*Information Theory, IEEE Transactions on* 42, no. 2 (1996): 566-578.

[2-15] Rogaway, Phillip. "Bucket hashing and its application to fast message authentication." In *Advances in Cryptology—CRYPT0'95*, pp. 29-42. Springer Berlin Heidelberg, 1995.

[2-16] Johansson, Thomas. "Bucket hashing with a small key size." In *Advances in Cryptology—EUROCRYPT'97*, pp. 149-162. Springer Berlin Heidelberg, 1997.

[2-17] Halevi, Shai, and Hugo Krawczyk. "MMH: Software message authentication in the Gbit/second rates." In *Fast Software Encryption*, pp. 172-189. Springer Berlin Heidelberg, 1997.

[2-18] Sarkar, Palash. "A new multi-linear universal hash family." *Designs, Codes and Cryptography* (2012): 1-17.

[2-19] Brassard, Gilles. "On computationally secure authentication tags requiring short secret shared keys." *Advances in Cryptology CRYPTO'82 Proceedings, Springer-Verlag* (1983): 79-86.

[2-20] Black Jr, John R. "Message authentication codes." PhD diss., UNIVERSITY OF CALIFORNIA, 2000.

[2-21] Nguyen, Long Hoang, and Andrew William Roscoe. "Short-output universal hash functions and their use in fast and secure message authentication." In*The proceeding of the 19th international workshop on fast software encryption FSE*. 2012.

[2-22] Kaps, J-P., Kaan Yuksel, and Berk Sunar. "Energy scalable universal hashing."*Computers, IEEE Transactions on* 54, no. 12 (2005): 1484-1495.

[2-23] Katz, Jonathan, and Yehuda Lindell. *Introduction to modern cryptography*. Chapman & Hall, 2008.

[2-24] Bellare, Mihir, Ran Canetti, and Hugo Krawczyk. "Keying hash functions for message authentication." In *Advances in Cryptology—CRYPTO'96*, pp. 1-15. Springer Berlin Heidelberg, 1996.

[2-25] Katz, Jonathan, and Andrew Y. Lindell. "Aggregate message authentication codes." In *Topics in Cryptology–CT-RSA 2008*, pp. 155-169. Springer Berlin Heidelberg, 2008.

[2-26] Castelluccia, Claude, Einar Mykletun, and Gene Tsudik. "Efficient aggregation of encrypted data in wireless sensor networks." In *Mobile and Ubiquitous Systems: Networking and Services, 2005. MobiQuitous 2005. The Second Annual International Conference on*, pp. 109-117. IEEE, 2005.

[2-27] Iwasaki, Kazuhiko, and Tadahiko Nishimukai, "Aliasing Probability and Weight Distributions of Several Codes," *Systems and Computations in Japan*, vol 20, Issue 9, 1989.

[2-28] Pradhan, Dhiraj K., Sandeep K. Gupta, and Mark G. Karpovsky. "Aliasing probability for multiple input signature analyzer." *Computers, IEEE Transactions on* 39, no. 4 (1990): 586-591.

[2-29] Iwasaki, K., and F. Arakawa. "An analysis of the aliasing probability of multiple-input signature registers in the case of a $2^m$-ary symmetric channel." *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 9, no. 4 (1990): 427-438.

[2-30] Daehn, W., T. W. Williams, and K. D. Wagner. "Aliasing errors in linear automata used as multiple-input signature analyzers." *IBM Journal of Research and Development* 34, no. 2.3 (1990): 363-380.

[2-31] Khan, Omar I., and Michael L. Bushnell. "Aliasing Analysis of Spectral Statistical Response Compaction Techniques." In *Proceedings of the 19th International Conference on VLSI Design held jointly with 5th International Conference on Embedded Systems Design*, pp. 801-806. IEEE Computer Society, 2006.

[2-32] Pradhan, Dhiraj K., and Sandeep K. Gupta. "A new framework for designing and analyzing BIST techniques and zero aliasing compression." *Computers, IEEE Transactions on* 40, no. 6 (1991): 743-763.

[2-33] Karpovsky, M. G., S. K. Gupta, and D. K. Pradhan. "Aliasing and Diagnosis Probability in MISR and STUMPS Using a General Error Model." In *Test Conference, 1991, Proceedings., International*, p. 828. IEEE, 1991.

[2-34] Morii, Masakatu, and Kazuhiko Iwasaki. "A note on aliasing probability for multiple input signature analyzer." *Computers, IEEE Transactions on* 42, no. 9 (1993): 1152.

[2-35] Iwasaki, Kazuhiko, F. E. N. G. Shou-Ping, Toru FUJIWARA, and Tadao KASAMI. "Comparison of Aliasing Probability for Multiple MISRs and *M*-Stage MISRs with *m* Inputs." *IEICE TRANSACTIONS on Information and Systems* 75, no. 6 (1992): 835-841.

[2-36] Pilarski, Slawomir, Tiko Kameda, and Andre Ivanov. "Sequential faults and aliasing." *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 12, no. 7 (1993): 1068-1074.

[2-37] Lempel, Mody, and Sandeep K. Gupta. "Zero-aliasing for modeled faults."*Computers, IEEE Transactions on* 44, no. 11 (1995): 1283-1295.

[2-38] Damiani, Maurizio, P. I. E. R. O. Olivo, M. I. C. H. E. L. E. Favalli, S. I. L. V. I. A. Ercolani, and B. Ricco. "Aliasing in signature analysis testing with multiple input shift registers." *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on* 9, no. 12 (1990): 1344-1353.

[2-39] Hadjicostis, Christoforos N. "Aliasing probability calculations for arbitrary compaction under independently selected random test vectors." *Computers, IEEE Transactions on* 54, no. 12 (2005): 1614-1627.

[2-40] Jianwu, Zhao, Shi Yibing, and Li Yanjun. "Aliasing Probability for Single Input Linear Feedback Signature Registers." In *Electronic Measurement and Instruments, 2007. ICEMI '07. 8th International Conference on,* pp. 995-999. IEEE, 2007.

[2-41] Edirisooriya, Geetani, and John P. Robinson. "Aliasing probability in multiple input linear signature automata for q-ary symmetric errors." In *Computer Design: VLSI in Computers and Processors, 1991. ICCD'91. Proceedings, 1991 IEEE International Conference on,* pp. 352-355. IEEE, 1991.

[2-42] Savir, Jacob. "Reducing the MISR size." *Computers, IEEE Transactions on* 45, no. 8 (1996): 930-938.

[2-43] Preneel, Bart. "Analysis and design of cryptographic hash functions." PhD diss., Katholieke Universiteit te Leuven, 1993.

[2-44] Vosoughi, Aida, and Rajendra Katti. "Fast Message Authentication Code for Multiple Messages with Provable Security." In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE,* pp. 1-5. IEEE, 2010.

[2-45] Sule, Rucha, Raj S. Katti, and Rajesh G. Kavasseri. "A variable length fast Message Authentication Code for secure communication in smart grids." In *Power and Energy Society General Meeting, 2012 IEEE*, pp. 1-6. IEEE, 2012.

[2-46] Manteena, Rajender. “A VHDL implementation of the Advanced Encryption Standard-Rijndael Algorithm” Master’s Thesis., University of South Florida, 2004.

# CHAPTER 3. A VARIABLE LENGTH FAST MESSAGE AUTHENTICATION CODE FOR SECURE COMMUNICATION IN SMART GRIDS



## 3.1. Abstract

We propose a variable length Message Authentication Code (MAC) scheme for secure communication between Automated Metering Interface (AMI) devices and collector nodes in the smart grid. We prove the security of this scheme and analyze its performance with respect to three attributes namely: (i) communication overhead, (ii) verification delay and (iii) memory usage. The proposed scheme reduces the time for verification by at least two orders compared to existing hash based authentication protocols. The scheme thus provides an efficient solution to support high frequency exchange of large volume messages.

## 3.2. Introduction

The smart grid will feature an electric grid that is closely intertwined with the communication (or information) network. The purpose of the communication network is to allow two-way interactions between the end-user and electricity service provider. Two-way communication in distribution grids is achieved through an Automated Meter Interface (AMI) (or “smart meter”). The AMIs can monitor, record and transmit electrical variables from the customer while conveying command/control information to the customer. Outbound messages to

the MIs typically include information such as real (kW) and receive power (kVar), power-factor, voltage profiles (sag/swell events) and peak usage. Inbound messages to the AMIs include control/command requests from utilities to implement automated net-metering, remote service disconnect/reconnects; real-time electricity pricing to customers for active load management and demand response. To address security concerns with communication, the National Institute of Standards and Technology (NIST) recommends "mutual authentication" between AMIs and service providers [3-1].

Achieving mutual authentication in smart-grids is challenging given the large number (several hundreds or thousands within a service territory) of AMI devices, large message volumes and frequency of message exchanges. Traditional public-key infrastructure based schemes are not well suited for secure communication in the smart-grid because of: (i) increased communication burden (large key sizes which increase communication bandwidth), (ii) increased time for decryption/verification (which increase latency) and (iii) the limited computational abilities of AMIs. Thus there is a need to develop "lightweight" authentication schemes that do not overburden the system in terms of communication and computational requirements.

Cyber-security and power system communication requirements for the smart grid are discussed in [3-2], [3-3]. An efficient scheme (in terms of authentication delay and computational cost) for authentication in multicast (one to many) mode is presented in [3-4]. A lightweight scheme based on Diffie-Hellman key exchange protocol and a hash based message authentication code (HMAC) is proposed in [3-5]. In [3-5], HMAC with RIPEMD128 as the underlying hash function is used to perform authentication. RIPEMD is known to be insecure and it is better to use SHA1 or SHA2 instead. Another drawback with using HMAC is that its authentication speed limits message rates between the smart-meters and collector nodes. Using

the HAMC also limits the size of messages that can be exchanged without extra buffering or delays.

We propose a new low-cost high-speed variable length message authentication code (MAC) that uses a multiple input shift register (MISR) and one computation of AES to generate a MAC tag. Our method has a higher authentication speed and can hence withstand higher rate of message exchanges. Our method can also handle larger messages (for example - 40,320 bytes representing one month's data, [3-5]) without incurring any extra delay or extra buffering.

The rest of the paper is organized as follows. The structure of communication system is described in section 3.3. The proposed scheme is presented in section 3.4 and its security is proved in section 3.5. The performance of the proposed scheme is compared with existing methods in section 3.6 and conclusions are noted in section 3.7.

## 3.3. Communication Structure

A rough framework for the communication structure is shown in Figure 3.1. the arrangement is hierarchical, starting from AMI devices at the customer leading up to the Enterprise or utility. Here, an AMI device at every customer is denoted by $HAN_{GW}$ (Home Area Network Gate Way). At this level, the preferred communication system is Zigbee/mesh wireless. Groups of $HAN_{GW}$ communicate with $BAN_{GW}$ (Building Area Network Gate Way) which serves as the collector node where WiMAx is preferred communication medium. Groups of $NAN_{GW}$s (Neighborhood Area Network Gate Way) relay the information further to a higher collector node which may be an enterprise, utility provider, or a trusted third party. A survey of appropriate communication technologies for the smart grid can be found in [3-8].

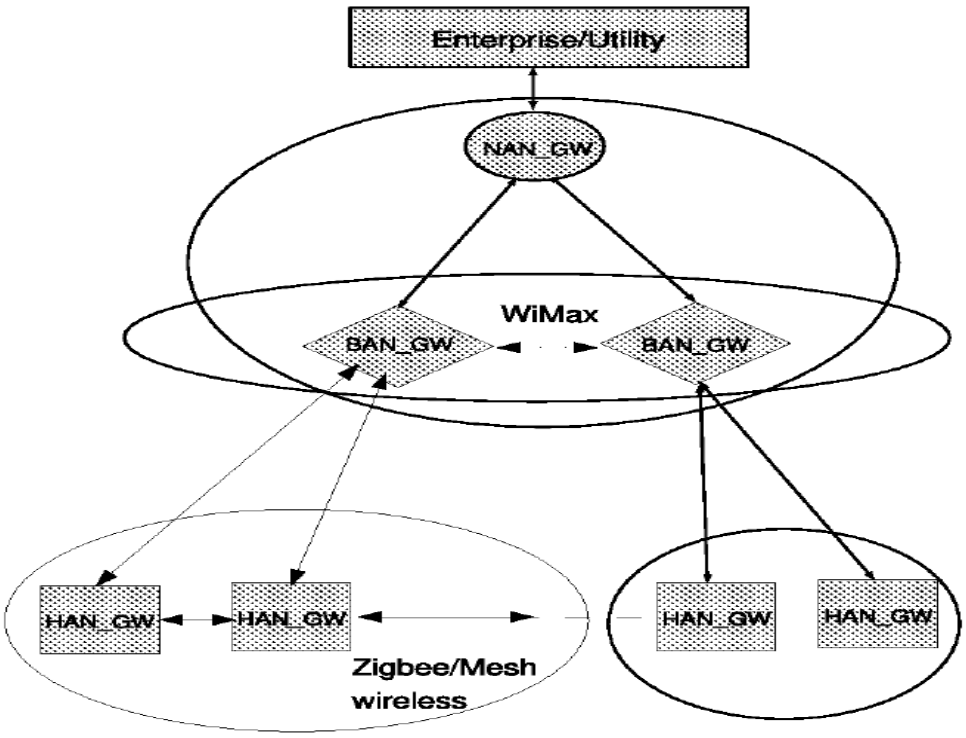


Figure 3.1. Communication Structure.

## 3.4. Proposed Scheme

We will follow the same protocol as in [3-5], however instead of using $HMAC_{K_i}$, we propose a new variable length MAC which will render the above protocol more efficient in terms of communication overhead, verification delay, and memory usage. We will start by defining Message Authentication Codes (MAC).

*Definition 3.4.1:*

Formally, a message authentication code (MAC) is a tuple of probabilistic polynomial-time algorithms (Gen, Mac, Vrfy) in which Gen is the key generation algorithm, Mac is the tag generation algorithm, and Vrfy is the verification algorithm. It is required that for every $n$ (a

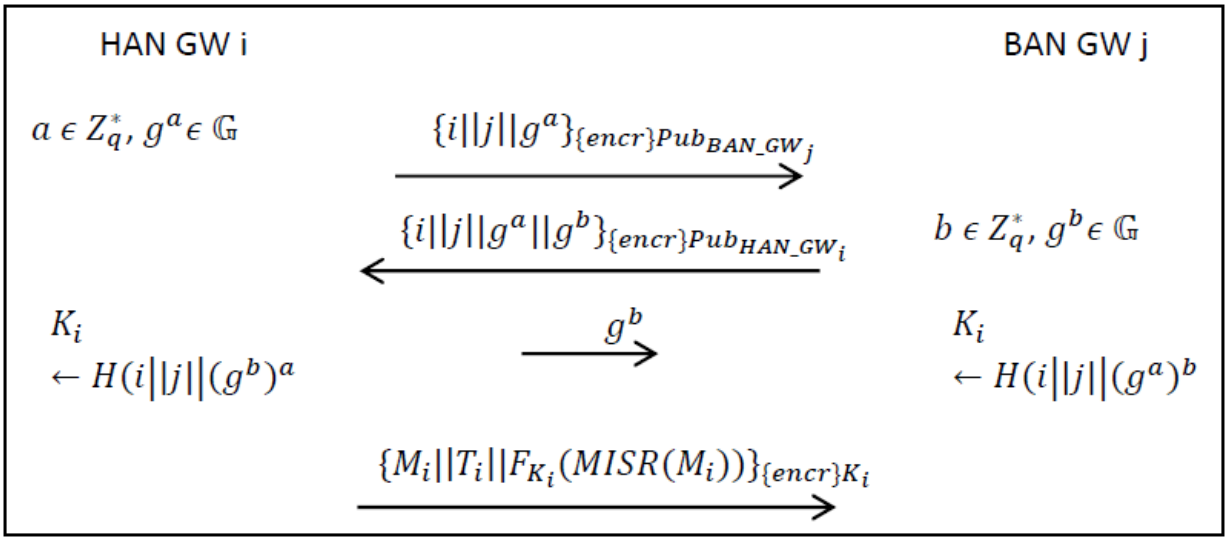


Figure 3.2. Proposed Scheme.

security parameter that usually specifies the length of key $k$), every $k$ output by Gen, and every $m$, it holds that $Vrfy\big(m, MAC_k(m)\big) = 1$ . If $\Pi = (Gen, Mac, Vrfy)$ is such that Mac is only defined for messages of a certain length, then the MAC scheme is said to be fixed-length, otherwise it is called variable-length MAC [3-6].

The proposed scheme is a variable-length MAC based on fixed-length MAC described in [3-7]. The scheme is as follows: Given any message $M \in \{0,1\}^*$of length $d$, check if $d \; mod \; (L) = 0$, where $L = 1024$. If not then pad it with $\lceil (d + \log_2 d)/(L) \rceil * L - d - \log_2 d$ bits of zeros and append $\log_2 d$ bits of length at the end. (Note that *d* has to be multiple of 128*8 = 1024.) Otherwise pad M with $L - \log_2 d$ zeros and add $\log_2 d$ bits of length at the end. We will call such a message $M^p = [M \; || \; \bar{0} \; ||(d)_b]$, where $||$ indicates concatenation. Figure 3.3 shows structure of $M^p$. Now parse $M^p$ into $L/8 = l = 128$ blocks each of size $s = |M^p|/l$, where , $|M^p|$ indicates length of $M^p$ in bits. These message blocks can then be input into MISR. The contents of flip-flops after s bits of $l$ messages have been input into MISR form a sort of remainder, $R$, resulting from division of $l$ messages by characteristic polynomial of MISR.

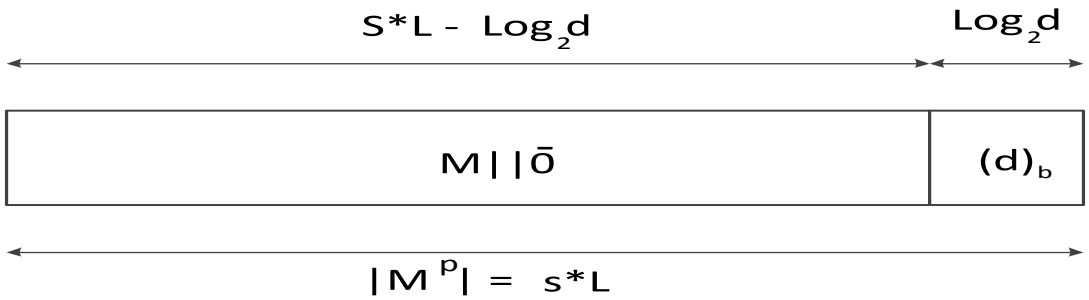

Figure 3.3. Structure of $M^p$.

The tag of $l$ messages can be computed as $F_k(R)$, where $F_k$ is a pseudorandom function. A pseudorandom function is defined below.

*Definition 3.4.2:*

Let F: $\{0,1\}^n \times \{0,1\}^n \rightarrow \{0,1\}^n$ be an efficient (polynomial-time), length-preserving, keyed function. We say that $F$ is a pseudorandom function if for all probabilistic polynomial-time distinguishers $D$, there exists a negligible function $negl(.)$ such that:

$$\left|(Pr\left[D^{F_k(.)}(n)\right] = 1) - (Pr\left[D^{f(.)}(n)\right] = 1)\right| \leq negl(n),$$

where $k \leftarrow \{0,1\}^n$ is chosen uniformly at random and $f$ is chosen uniformly at random from the set of functions mapping $n$-bit strings to $n$-bit strings.

*Remark 3.4.1:*

$F_k$ is chosen from one of $2^n$ distinct functions (one for each value of the $n$-bit key $k$). $f$ is chosen from the set of all $2^y$ functions with $n$-bit input and output, where $y = n2^n$. $D$ is given oracle access to some function (either $F_k$ or $f$, denoted $D^{F_k(.)}$ or $D^{f(.)}$) and its goal is to determine if this function is $F_k$ or $f$. $Pr\left[D^{F_k(.)}(n)\right] = 1$ denotes the fact that $D$ is successful when given

oracle access to $F_k$. If no probabilistic polynomial-time distinguisher *D*, can tell which function *D* has oracle access to, then $F_k$ is a pseudorandom function. Note that *D* can query the oracle function polynomial number of times. Thus, even if $x_1$ and $x_2$ differ in only a single bit $F_k(x_1)$ and $F_k(x_2)$ look completely uncorrelated. Block ciphers such as AES can be used as pseudorandom functions [3-6]. We will use AES128 for this purpose. Pseudorandom functions can be used as fixed length MAC, we will make use of this fact when we will sketch the proof of security for our MAC.

### 3.4.1. Construction $\pi$

- *Gen:* On input *n* (the security parameter), $K_i$ obtained in setup phase (see Figure 3.2), and a secret irreducible polynomial $g(x)$ over $GF(2)$ of degree *n*. We will fix $n = l$. Note that $g(x)$ is selected only once and hence the key is $(K_i, g(x))$.
- *Mac:* On input $K_i$, irreducible polynomial $g(x)$ over $GF(2)$ of degree *n*, and a message $M \in \{0,1\}^*$of length $d$, check if $d \bmod L = 0$. If not then pad it with $\lceil d/(L) \rceil * L - d - \log_2 d$ bits of zeros and append $\log_2 d$ bits of length at the end. Otherwise pad M with $L - \log_2 d$ zeros and add $\log_2 d$ bits of length at the end to get $M^p$. Parse $M^p$ into $l$ blocks each of length *s*, such that $M^p = (m_1, \ldots, m_n)$. Input all $m_i$'s into MISR with characteristic polynomial $g(x)$. Contents of the flip flops after *s* cycles is called $MISR(M) = R$, $(R \in \{0,1\}^n)$. Output the $tag\ t = F_k(R)$.
- *Vrfy:* Output 1 if and only if $t := F_k\big(MISR(M)\big)$.

## 3.5. Security of the Proposed Scheme

There is generally-accepted definition of security for message authentication codes. In simple words, no polynomial time adversary should be able to generate a valid tag for a message

which was not previously authenticated. Note that the adversary is allowed to request MAC tags for any messages of its choice. Toward the formal definition, consider the following experiment for a message authentication code $\Pi = (Gen, Mac, Vrfy)$, adversary $\mathcal{A}$, and security parameter $n$.

Experiment $Mac - forge_{A,\pi}(n)$:

1) Run $Gen(n)$ to obtain a random key $k$.

2) The adversary $\mathcal{A}$ is given oracle access to $MAC_k(.)$. The adversary eventually outputs a pair $(m, t)$. Let $Q$ denote the set of all queries that $\mathcal{A}$ asked its oracle.

3) The output of the experiment is defined to be 1 if and only if (1) $Vrfy_k(m, t) = 1$ and (2) $m \notin Q$.

*Definition 3.5.1 : Security Definition*:

A message authentication code $\Pi = (Gen, Mac, Vrfy)$ is said to be existentially unforgeable under an adaptive chosen message attack, or just secure, if for all probabilistic polynomial-time adversaries $\mathcal{A}$, there exists a negligible function $negl(.)$ such that $Pr[Mac - forge_{\mathcal{A},\pi}(n) = 1] \leq negl(n)$.

A negligible function is defined as following: A function $f$ is negligible if for every polynomial $p(.)$ there exists an $N$ such that for all integers $n > N$ it holds that $f(n) < 1/p(n)$.

The security of the proposed scheme is stated in the following theorem:

*Theorem 3.5.1:*

If $F_k(.)$ is a secure fixed length MAC and function MISR(.) is collision resistant then

construction Π is a variable length MAC, that is existentially unforgeable under adaptive chosen message attacks.

*Proof:*

The security of proposed MAC relies on the assumption that MISR are collision resistant and $F_k(.)$ is a secure fixed length MAC. Essentially this new MAC construction first compresses the given message $M$ to fixed length binary string $R$, and then applies fixed length MAC $F_k(R)$. Assume towards contradiction that there exists a probabilistic polynomial time adversary $\mathcal{A}$ attacking MAC scheme that forges valid tag on a new message with non-negligible probability. $\mathcal{A}$ is given access to a MAC oracle, that it can query for a tag on any message of its choice. Let $M^*$ denote the message for which $\mathcal{A}$ produces its forgery, and let $Q$ denote the set of queries made by $\mathcal{A}$ to its MAC oracle (i.e., the set of messages for which it obtained a MAC tag). We can assume without loss of generality that $M^* \notin Q$(since $\mathcal{A}$ cannot succeed otherwise). There are two possible cases:

1. Case 1: ∃ a message $M \in Q$ *such that* $MISR(M) = MISR(M^*)$.

   In this case, the MAC tag for $M$ is equal to the MAC tag for $M^*$ and so clearly $\mathcal{A}$ can successfully forge a valid tag on $M^*$. However, this case contradicts the assumption that $MISR(.)$ is collision resistant because $\mathcal{A}$ could find distinct $M$ and $M^*$ for which $MISR(M) = MISR(M^*)$. Hence this case cannot be valid as long as $MISR(.)$ is collision resistant (It can be proved that $MISR(.)$ is collision resistant but it is out of scope of this paper).

2. Case 2: *for every message* $M \in Q$ *it holds that* $MISR(M) \neq MISR(M^*)$.

   Define $Q' = \{MISR(M) | M \in Q\}$. The important observation here is that $M^*$ is such that $MISR(M^*) \notin Q'$. In this case then $\mathcal{A}$ is forging a valid message on the new message

$MISR(M^*)$ with respect to fixed length message authentication code $F_k$. This contradicts assumption that $F_k$ is secure fixed length MAC rendering this case invalid.

Hence by contradiction we can say that $\mathcal{A}$ cannot be successful in outputting a valid forgery given access to MAC oracle $Q$. Therefore $Pr[Mac - forge_{\mathcal{A},\pi}(n) = 1] \leq negl(n)$. Hence given MAC scheme is existentially unforgeable under chosen message attacks. [3-2]

## 3.6. Results

We will compare the results when the HMAC scheme in [3-5] is replaced with proposed variable length MAC scheme. Hence we basically show the improvement in the performance when HMAC is replaced with our new scheme. The performance of scheme is compared with respect to:

1) Communication overhead
2) Message Decryption and Verification Delay
3) Memory Usage

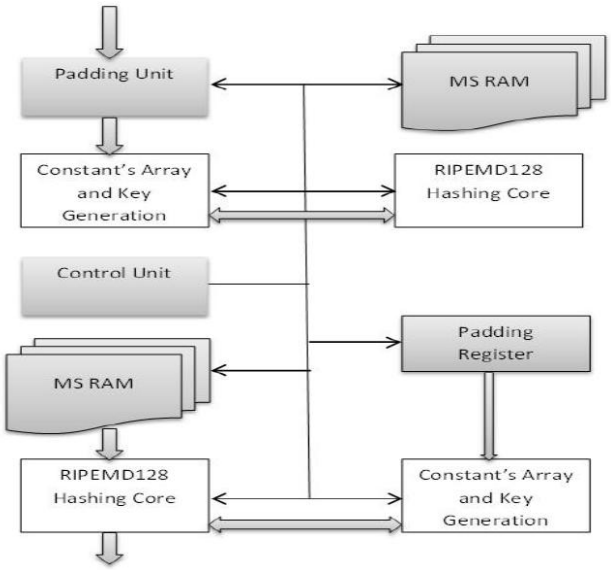


Figure 3.4. HMAC Scheme.

Before comparing both schemes we should go into details of the algorithms used for verification in [3-5] and our new scheme. The HMAC construction is as shown in Figure 3.4.

[3-5] are using RIPEMD128 as the underlying collision resistant hashing function. However, RIPEMD128 is not considered as a secure collision resistant hash and hence its use is questionable. HMAC is variable length MAC and it produces a tag of constant length which will be 128bits in this case. We will be using 128 bit MISR which will run for at least 8 cycles as per our construction and then we will feed the compressed output to AES128 to give us 128 bit tag. We will also have constant tag length out of our variable length MAC. The performance is verified as follows:

### 3.6.1. Communication Overhead

As mentioned earlier, the most commonly used message that is 32bytes. HMAC produces constant tag length of 16 bytes for any size of message. The message header is 50 byes. Hence the total communication overhead at $BAN_{GW}$ will be 98 bytes. Our scheme will also produce a 16 byte tag on 32 byte message. Hence both the schemes will perform the same in terms of communication overhead as shown in Figure 3.5. With increase in message size communication overhead will also increase but will still be within 30kBytes for message of length 150Bytes. Hence the proposed scheme is as good as the one in [3-5] in terms of communication overhead. For comparison, the overhead (highest) with Elliptic Curve Digital Signature Algorithm (ECDSA) is also shown.

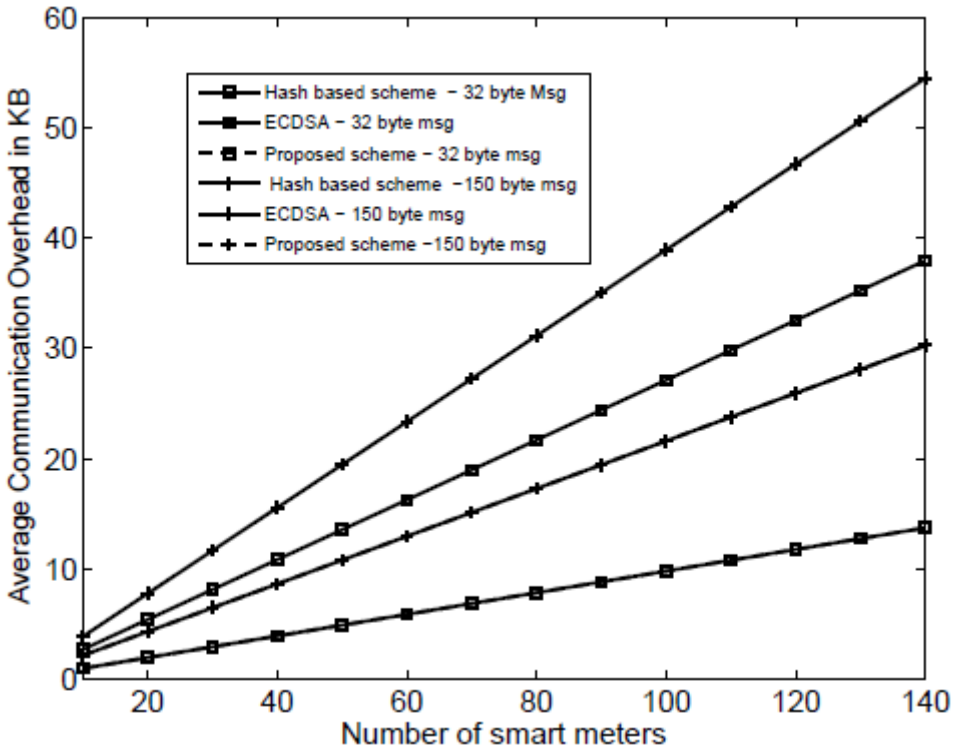


Figure 3.5. Communication Overhead.

### 3.6.2. Verification and Decryption Delay

Both the schemes use AES128 for decryption. Hence we will compare the schemes only for verification delay. For message of length 32bytes, their scheme will perform 5 operations of hash core that is RIPEMD128. RIPEMD 128 core requires 592 cycles to compute hash [3-9]. Hence each hash will require (592/160MHz) = 3.7uS. Hence total time required for tag computation will be at least of 5 hash core operations = 18.5uS. This delay will increase with increase in message size. In addition to it there will be some delay associated with key establishment, padding etc. As observed from graphs of [3-5], average delay for every $BAN_{GW}$ per $HAN_{GW}$ is approximately 80msec. Out of which 18.5uS will be introduced due to HMAC. However our scheme will need 8 cycles for computing MISR(M) and $25 * 16 = 400$ cycles for computing AES128 using Rijndael [3-10]. Hence total time of (408/160MHz) = 2.55uSec. Our

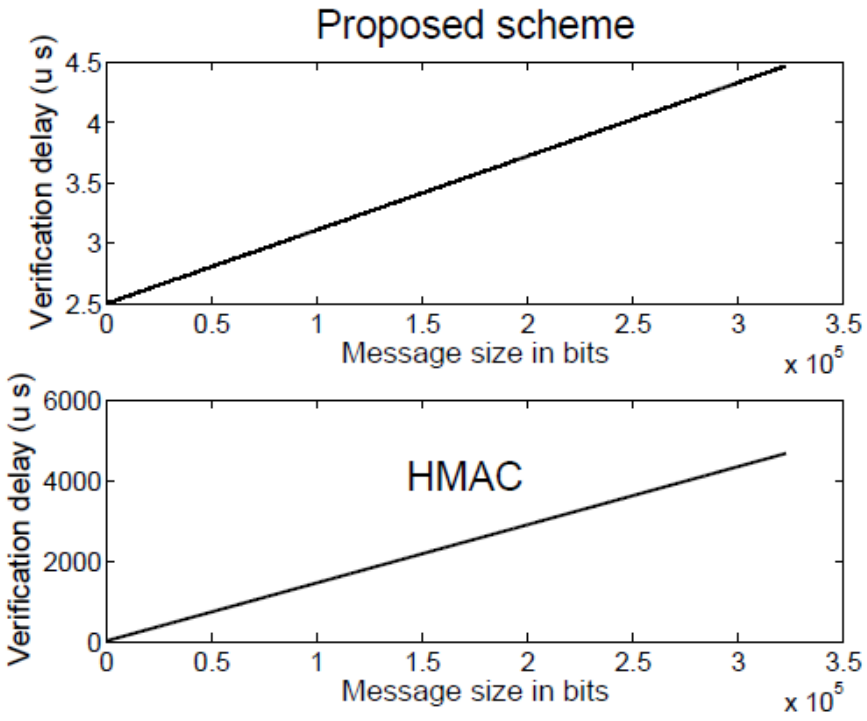


Figure 3.6. Verification Overhead.

scheme shows seven times better performance for 32 bytes of data. It will continue to have constant verification time of 2.6uSec as long as length of message is less than 128 bytes.

For messages of length between 128-256 bytes, it will require 416 cycles and hence will require total verification delay of 2.6uSec. As can be seen from Figure 3.6, our scheme can support wide range of message sizes with verification delay less than 5uSec which is quite small. Hence we claim that replacing HMAC will not only decrease total verification delay but also supports wide range of message sizes with drastically small delays. For 32 byte data our scheme will replace the delay of 18.5uSec + (delay for padding in uSec) with 2.55uSec. However with higher message sizes our scheme will prove more promising. In conditions where $HAN_{GW}$s are sending scheduled one month data, it might increase verification delay at $BAN_{GW}$ drastically whereas our scheme will not add much delay and $BAN_{GW}$ will almost be unaffected. Figure 3.7

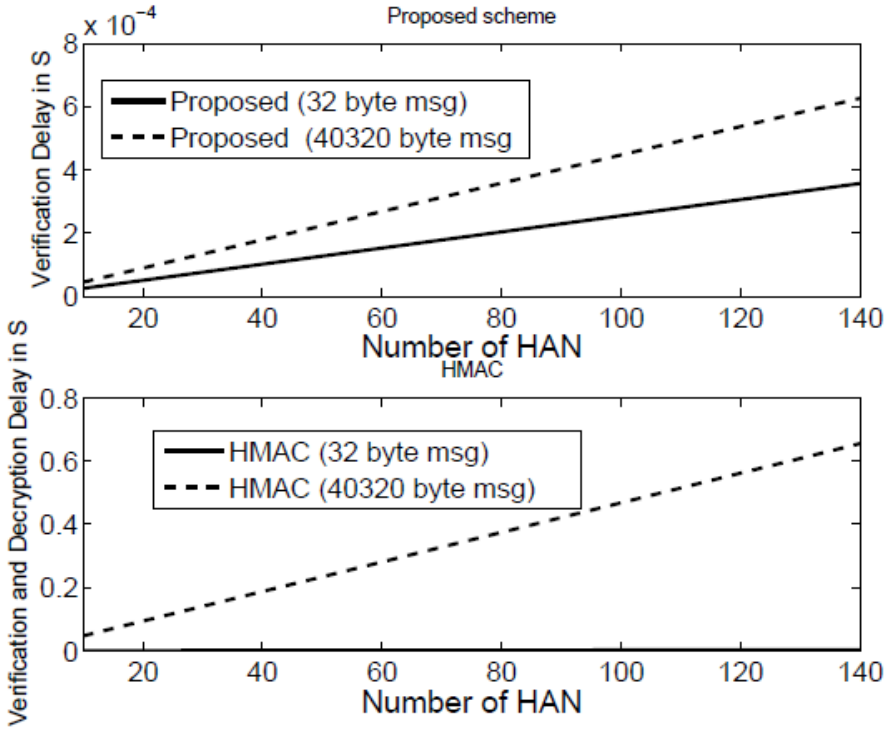


Figure 3.7. Verification and Decryption Delay.

shows comparison of verification delay vs number of $HAN_{GW}$. Our scheme contributes to a delay of 0.357 mSec for 140 HANs, whereas HMAC based scheme contributes delay of 2.590 mSec for 140 HANs. With 40230 bytes of monthly data, it will take approximately 0.6mSec whereas HMAC will require 0.6sec. Thus the proposed scheme will perform 1000 times faster.

### 3.6.3. Memory Usage

From Figure 6 in [3-5], to process 1 message of 32 bytes $BAN_{GW}$ takes approximately 80msec. Hence to process 125 messages it will take approximately 10sec. Let message generation interval $\Delta$ be 10 sec. Hence, as long as message rate is below 125, $BAN_{GW}$ does not need to buffer messages for execution of algorithms. But as soon as message rate goes above 125, it will require more memory for buffering new messages. For example if message rate is 150, $BAN_{GW}$ has to buffer 50 messages for first 10 sec, then 75 messages for next 10 sec and so

on. This size will keep on increasing. However, as our scheme is faster than hash function based approach, we will be able to support higher message rate than that of conventional HMAC based scheme. Now suppose one of the $HAN_{GW}$ is sending 40320 bytes monthly data. In this case HMAC verification delay will be (4*592)/160M + (((40*8k)/256)*592)/160M = 4.67mSec ≈ 5ms. Hence the total delay to process message will become 85mSec, with which $BAN_{GW}$ can support 117 messages/ Δ. However with our scheme it will be 124. Hence replacing HMAC with our scheme will prove an efficient solution for messages of large size.

### 3.6.4. Discussion

As emphasized in the above section, our scheme is an alternative for traditional heavy weight MAC. [3-5] uses RIPEMD128 which is not considered to be secure. Even if we replace it with any secure collision resistant hash like SHA1 or SHA2, no scheme will perform as fast as our scheme on messages of very large size. Being a variable length MAC there is no restriction on message size. Also with our scheme it is possible to combine several messages together to produce a tag of constant length and fairly constant verification time. This might prove useful when the message rate is high, instead of sending several messages per Δ, $HAN_{GW}$ can wait and combine several messages together to have single tag and send it across to $BAN_{GW}$. For example if message rate is 250, then $HAN_{GW}$ can combine two messages together and produce one tag for them, hence total messages coming to $BAN_{GW}$ will still be 125, in this manner we can provide efficient memory usage. Hence we claim that our scheme will prove to be a good option for lightweight authentication of messages of variable size with fairly constant authentication and verification time.

## 3.7. Conclusion

This paper presents a new message authentication scheme that is faster than best scheme proposed recently [3-5] for the smart grid. Our scheme uses a multiple input shift register (MISR) to compress the message that needs to be authenticated. The compressed output is then input to a pseudorandom function such as AES128. The output of AES128 is then the tag in the authentication scheme. Since there are efficient methods to implement both AES and MISRs in hardware, our scheme results in very low verification times. This translates to several advantages in the smart-grid. Firstly, the rate at which a smart-meter in a home (also called HAN) can generate messages without extra buffering goes up resulting in better memory usage in the HAN. Secondly, there is an increase in the message length that can be authenticated without buffering. This can enable more information exchange between HAN and the BAN.

## 3.8. References


[3-1] "Guidelines for Smart Grid Cyber Security: Vol. 2, Privacy and the Smart Grid", *The Smart Grid Interoperability Panel- Cyber Security Working Group*, August 2010.

[3-2] Ericsson, Göran N. "Cyber security and power system communication—essential parts of a smart grid infrastructure." *Power Delivery, IEEE Transactions on* 25, no. 3 (2010): 1501-1507.

[3-3] Hauser, Carl H., David E. Bakken, Ioanna Dionysiou, K. Harald Gjermundrod, Venkata Irava, Joel Helkey, and Anjan Bose. "Security, trust, and QoS in next-generation control and communication for large power systems." *International Journal of Critical Infrastructures* 4, no. 1 (2008): 3-16.

[3-4] Li, Qinghua, and Guohong Cao. "Multicast authentication in the smart grid with one-time signature." *Smart Grid, IEEE Transactions on* 2, no. 4 (2011): 686-696.

[3-5] Fouda, Mostafa M., Zubair Md Fadlullah, Nei Kato, Rongxing Lu, and Xuemin Shen. "A lightweight message authentication scheme for smart grid communications." *Smart Grid, IEEE Transactions on* 2, no. 4 (2011): 675-685.

[3-6] Katz, Jonathan, and Yehuda Lindell. *Introduction to modern cryptography*. Chapman & Hall, 2008.

[3-7] Vosoughi, Aida, and Rajendra Katti. "Fast Message Authentication Code for Multiple Messages with Provable Security." In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pp. 1-5. IEEE, 2010.

[3-8] Akyol, B. A., Harold Kirkham, S. Clements, and M. Hadley. "A survey of wireless communications for the electric power system." *Prepared for the US Department of Energy* (2010).

# CHAPTER 4. MULTICAST AUTHENTICATION IN THE SMART GRID WITH ONE-TIME SIGNATURES FROM SIGMA PROTOCOLS

This paper has been submitted for publication to IEEE Transactions on Smart Grid. The authors of the paper are Rucha S. Sule, Rajendra S Katti, and Rajesh Kavasseri.

## 4.1. Abstract

Multicast authentication is challenging in the smart grid given the unique constraints of communication bandwidth, computation time, and limited computational resources of field devices. We propose here, for the first time, the use of one-time signatures (OTS) from sigma protocols for multicast authentication in the smart grid. As a zero knowledge proof, sigma protocols interactively establish the truth of a statement without revealing its contents; thus providing a powerful paradigm for authentication. When compared with the currently best known OTS scheme -Tunable Signing and Verification [4-1], the proposed scheme yields a dramatic reduction in: signing cost (three orders), pre-computation cost (two orders), and storage overhead (two orders) at a very modest increase (four fold) in signature size. The scheme thus efficiently enables several multicast applications in the smart grid environment ranging from the distribution through sub-transmission and bulk power systems with resource-constrained field devices.

## 4.2. Introduction

The smart grid incorporates a communication network closely intertwined with the electric grid to facilitate two-way communication. The nature of communication (type of message and message transmission mode) depends upon the specific application and the grid

level-distribution, sub-transmission or bulk transmission. For example, at the distribution level, communication between an end-user and electricity service provider can be achieved through Automated Meter Interface (AMI) (or "smart meters"). The AMIs can monitor, record and transmit electrical variables from the customer (outbound) while relaying command/control information (inbound) to the customer. Outbound messages from AMIs typically include information such as real (kW) and reactive power (kVar), power-factor, voltage profiles (sag/swell events) and peak usage. Inbound messages to the AMIs include control/command requests from utilities to implement automated net-metering, remote-service disconnects/reconnects; real-time electricity pricing to customers for active load management and demand response.

Message transmission in the smart grid can be realized in unicast or multicast mode depending on the application. In unicast mode, a message is exclusive or unique to a specific user. Unicast is useful for control or command type messages used for direct load control in demand side management (DSM) programs such as those implemented by Florida Power and Light Company [4-2]. For example, in residential load control programs, a utility may directly intervene to control a customer's load through selective appliance switch on/off commands. In this context, the exclusivity of the contract between the supplier (utility) and customer necessitates unicast communication. However, certain applications require the transmission of a message that is shared (or common) to multiple users. Such messages are transmitted as multicast because unicast may be too expensive (in terms of computational resources) and therefore inefficient in such cases. For example, a utility may wish to proclaim electricity prices to its list of subscribed customers, who may then initiate a load scheduling program beneficial to them; or utilities may encourage charging of Plug-in-Hybrid-Electric Vehicles when generation

from renewable resources is plentiful or cheap. In the bulk power system, control centers may initiate System Integrity Protection Schemes (SIPS) to mitigate or limit the propagation of major disturbances in the system [4-3].

Given the sensitivity of control/command messages in multicast, authentication is of crucial importance failing which malicious parties may gain unauthorized access, forge messages or mount a replay attack with potentially catastrophic consequences. The National Institute of Standards and Technology (NIST) recommends "mutual authentication" between [4-4] to verify that a message delivered to an entity indeed originates from its intended sender.

However, achieving mutual authentication in smart-grids is challenging given the large number (several hundreds or thousands within a service territory) in the case of AMI devices, large message volumes and frequency of message exchanges. Traditional public-key infrastructure based digital signature schemes (such as RSA) cannot be adapted for secure communication in the smart-grid because of: (i) increased communication burden (large key sizes which increase communication bandwidth), (ii) increased time for decryption/verification (which increase latency) and (iii) the limited computational capabilities of AMIs and other field devices.

A promising solution to broadcast (and thus multicast) authentication applications is possible through one-time signatures (OTS). An OTS scheme [4-5] and [4-6] generates one digital signature based on a cryptographically secure one-way function without trapdoors for several messages that are multicast. This enables the use of a single key to sign several messages. While OTS schemes are generally fast, the drawback is a large signature size which limits their direct application in the smart-grid. The Bins and Balls (BiBa) scheme [4-7] provides an improvement by reducing the signature size, however, at the expense of time required to sign a

message. A significant reduction over the BiBa scheme is achieved by HORS (Hash to Obtain Random Subset) [4-8] which reduces the signing overhead and therefore HORS has been used for adaption in several multicast authentication applications ([4-15], [4-16], and [4-17]). Many others as [4-18], [4-19], and [4-20] address drawbacks of existing OTS schemes and suggest improvements to maintain smaller signature sizes while achieving fewer computations.

As pointed out in [4-1], HORS is not well suited for smart grid applications mainly because the scheme requires large public key sizes that increase storage requirements on resource constrained field devices. However, contrary to what is noted in [4-1]; the signature size of HORS (130 bytes) is not actually a serious concern for wide area protection applications. A Phasor Measurement Unit (PMU) simply records time synchronized phasor (voltage and current) and frequency measurements in accordance with IEEE C37.118.2 at system buses. To minimize communication bandwidth, PMU data are first sent to Phasor Data Concentrators (PDC). Data from the PDC is then routed to the regional control center via a security gateway and utility (local) control centers if necessary. The regional area control center (and not the PMU) is thus the primary entity that initiates wide area protection and control measures or SIPS which requires multicast of control/command data. As such the PMU is not expected to multicast recorded data to neighboring PMUs.

To address the primary drawbacks of HORS, [4-1] proposes a hybrid method called Tunable Signing and Verification (TSV). By adding one more level of restriction on signature generation and verification they improve the security by a factor of $k!$, where $k$ is the signature size. Therefore the same security can be achieved with smaller signature sizes. However, this is attained at the expense of increased computations at sender or receiver. Though they present a heuristic solution for flexible allocation of computations to the sender or receiver based on

availability of resources, the amount of computations required are extensive. However, in many smart grid applications, receivers of multicast messages like home appliances, or field devices, for example are highly resource constrained. In these cases, it is highly desirable to minimize the computations at the sending and receiving end. Hence, the need for better one-time signature schemes that will require fewer resources at the receiver, with modest signature sizes and low sender computations is both urgent and important in the smart-grid environment.

Therefore, we suggest an entirely new approach- that of sigma protocols for computing one-time signatures. While they have been traditionally used in applications like e-cash, e-voting, and e-credentials, their use in smart grids has not been explored yet. A sigma protocol is an interactive three move protocol between a prover $P$ and verifier $V$ to establish the veracity of a statement without explicitly revealing the contents. This makes it fundamentally appealing where authentication is required, such as in the smart-grid.

Here, we utilize an OTS through a witness hiding sigma protocol, namely the Okamoto protocol. Security of this OTS scheme is same as security of solving discrete log problem. We compare performance of sigma protocol based one-time signatures with TSV analytically and show that sigma protocol based OTS significantly reduce receiver computations as compared to both HORS as well as TSV. Moreover, the receiver does not have to store verification keys of length proportional to message size. We claim that our scheme will perform two orders better in terms of signing and pre-computation cost as well as key lengths. The signature length is constant and will be maximum 2kb. This can be reduced to minimum of 32bytes at the cost of reduced security. But 2kb signature size is acceptable as compared to message sizes of 80kb. Hence, our scheme will provide an economical solution for multicast authentication in smart grids.

The rest of the paper is structured as follows: Section 4.3 provides an introduction to sigma protocols based one-time signatures and presents protocol for smart grid applications. Section 4.4 provides brief overview of the authentication scheme in [4-1]; Section 4.5 presents comparative results and conclusions are noted in Section 4.6.

## 4.3. Sigma-Protocols and One-Time Signatures

In cryptographic protocols, zero-knowledge proofs ensure that malicious parties do not cheat. Zero-knowledge proofs are considered to be an expensive way of enforcing honest behavior. However, for some languages, zero-knowledge proofs are a very efficient way to prove honest behavior. For more information on zero knowledge see chapter 4 in [4-9]. In the following sections we will discuss the background information needed for deriving one time signature from sigma protocols.

### 4.3.1. Background

Let $R \subset \{0,1\}^* \times \{0,1\}^*$ be a binary relation with the restriction that if $(x, w) \in R$, then the length of $w$ (called the witness) is at most $p(|x|)$, where $p(.)$ is some polynomial and $|x|$ is the length of $x$. Define $L_R$ to be the set of inputs $x$ for which there exists a $w$ such that $(x, w) \in R$. Sigma-protocols as defined in [4-10] are three-move protocols between a prover, $P$, and verifier, $V$, in which $P$ and $V$ have a common input $x$. $P$ tries to prove to $V$ that either $x$ belongs to language $L_R$ or it knows a $w$ such that $(x, w) \in R$. The protocol template is shown below.

*Protocol 4.3.1.1: Sigma-Protocol Template for a relation R:*

- *Common Input:* $P$ and $V$ get $x$.
- *Private Input:* $P$ has a value $w$ such that $(x, w) \in R$.

- *The protocol:*

    1. $P$ sends $V$ a message $a$ with $r$ as randomness used to generate $a$.
    2. $V$ sends $P$ a random $N$-bit string $e$ which is called the challenge.
    3. $P$ sends a reply $z$. $V$ decides to accept or reject based on the values it has seen, namely $(x, a, e, z)$ which is called the conversation of the protocol.

$P$ and $V$ are probabilistic polynomial time algorithms with $P$'s only advantage being that it knows the witness $w$. If $V$ accepts then it is convinced that $P$ knows a $w$ such that $(x, w) \in R$. Moreover, if $V$ is honest it does not learn any more information than $(x, w) \in R$. This implies that the sigma-protocol is honest verifier zero knowledge. Note that the relation $R$ is such that it is hard to compute $w$ from $x$, otherwise $V$ could compute $w$ by itself and it does not need to be convinced that $P$ knows $w$. We now define sigma-protocols formally.

*Definition 4.3.1.1:*

A protocol $\pi$ is a sigma-protocol for relation $R$ if it is a three-round protocol of the form in Protocol 4.3.1.1 and the following requirements hold:

1. *Completeness:* If $P$ and $V$ follow the protocol on input $x$ and private input $w$ to $P$ where $(x, w) \in R$, then $V$ always accepts.
2. *Special Soundness:* There exists a polynomial-time algorithm $A$ that given any $x$ and any pair of accepting transcripts $(a, e, z)$, $(a, e', z')$ for $x$, where $e \neq e'$, outputs $w$ such that $(x, w) \in R$.
3. *Special honest verifier zero knowledge:* There exists a probabilistic polynomial time simulator $M$, which on input $x$ and $e$ outputs a transcript of the form $(a, e, z)$ with the same probability distribution as transcripts between the honest $P$ and $V$ on common input $x$.

In [4-10] it is shown that a sigma-protocol can be converted to zero-knowledge proofs or zero-knowledge proofs of knowledge using commitment schemes. In this paper we use sigma-protocols that are witness indistinguishable and witness hiding to construct efficient one-time signature schemes [4-11].

*Definition 4.3.1.2:*

A sigma-protocol is witness indistinguishable if the following two conditions are satisfied.

1. There are $q$ different witnesses that all satisfy the condition for acceptance in step 3 of Protocol 4.3.1.1.
2. No matter what a cheating verifier $V^*$ does, the protocol conversation gives no information on which of the $q$ witnesses is known by the prover.

*Definition 4.3.1.3:*

A sigma-protocol is witness hiding if no probabilistic polynomial time algorithm $A$, who sees the common input and talks to the prover can output a valid witness, $w$, with non-negligible probability.

A one-time signature scheme can be obtained from a witness-hiding sigma-protocol as follows. Assume that a relation $R$ has an instance generator. An instance generator is a probabilistic polynomial time algorithm $G$ which on input $1^k$ $(k = |x|)$ outputs a pair $(x, w) \in R$. A one-time signature scheme is defined below.

*Definition 4.3.1.4:*

A one-time signature scheme consists of three probabilistic polynomial time algorithms $(Gen, Sign, Vrfy)$.

- *Gen:* $Gen$ is an algorithm that is given input $1^n$ ($n$ is the security parameter) and

generates the key $(sk, vk)$ ($sk$ is signing key and is the signer's secret and $vk$, is the verification key that is known to all).

- *Sign:* $Sign$ is the signing algorithm that takes as input the message $m$ that needs to be signed and $sk$ and outputs the signature $\sigma = Sign(sk, m)$.
- *Vrfy:* $Vrfy$ is the verification algorithm that takes as input the received message $m$ and $vk$ and outputs 1 or 0 to indicate that the verification was successful or not. $Vrfy(vk, m) = 1$ or $0$.

*Definition 4.3.1.5:*

The one-time signature scheme is said to be secure if an adversary having seen at most one valid signature on one message cannot efficiently come up with valid signature on another message. [4-11]

The signer of a one-time signature uses an instance-generator to select a random $(x, w)$. Next the signer generates a first message $a$ of the sigma-protocol (here the signer acts like the prover and the receiver acts like the verifier in a sigma-protocol). The verification key is $vk = (x, a)$ and the signing key is $sk = (w, r)$. Note that $x$ is the common input to the sigma-protocol. The message, $m$ to be signed is taken to be the challenge $e$ in the sigma-protocol. The one-time signature is the reply $z$ in the sigma-protocol such that $(x, a, m, z)$ is an accepting conversation of the sigma-protocol. Thus, $\sigma = z$. The verification algorithm is the condition in the protocol that checks if a conversation $(x, a, m, z)$ is acceptable or not (performed in step 3 of Protocol 4.3.1.1).

This one-time signature scheme is secure according to Definition 4.3.1.5. Suppose that there exists an adversary that can compute a valid signature on a different message, given one valid signature. Then the adversary has two different signatures $z$ and $z$' for the same $vk =$

$(x, a)$. This implies that the adversary has two different conversations for the underlying sigma-protocol $(x, a, m, z)$ and $(x, a, m', z')$ (note $m \neq m'$). Therefore from the special soundness property of the protocol the adversary can compute $w$. However this contradicts the witness hiding property of the sigma-protocol. Thus there exists no adversary that can efficiently compute a valid signature on different messages, given one valid signature. The above one-time signature scheme is therefore secure.

We consider an example one-time signature scheme that is based on the Okamoto protocol that is a sigma-protocol. Let $G_q$ be a group of prime order $q$, with generators $g_1$ and $g_2$, set in such a way that no one can efficiently compute, $x$, such that $g_1 = g_2^x$. The Okamoto protocol is a sigma-protocol based on the relation R = $\{(h, (w_1, w_2)) \mid h = g_1^{w_1} g_2^{w_2}\}$. The Okamoto protocol proceeds as follows:

*Protocol 4.3.1.2: The Okamoto Protocol.*

- *Common Input: P* and *V* get $h = g_1^{w_1} g_2^{w_2}$.
- *Private Input: P* has a value $(w_1, w_2)$ such that $(h, (w_1, w_2)) \in R$.
- *The protocol:*
  1. $P$ sends $V$ a message $a = g_1^{v_1} g_2^{v_2}$, where $r = (v_1, v_2)$ is chosen uniformly at random from $Z_q$.
  2. $V$ sends $P$ a random $N$-bit string e in $Z_q$, which is called the challenge.
  3. $P$ computes $z_1 = ew_1 + v_1 \, mod \, q$, $z_2 = ew_2 + v_2 \, mod \, q$ and sends $z = (z_1, z_2)$ to $V$. $V$ accepts iff $g_1^{z_1} g_2^{z_2} = ah^e$.

A one-time signature scheme based on the Okamoto protocol is given below.

*One-time signature scheme based on the Okamoto protocol:*

1. *Gen:* The signing key and the verification key are: $sk = \big((w_1, w_2), (v_1, v_2)\big)$, $vk = h,a= (g1\,w1\,g2\,w2, g1\,v1\,g2\,v2)$.
2. *Sign:* The signature on a message $m$ is computed as $\sigma = (z_1, z_2) = (mw_1 + v1 \ mod\ q, mw2+v2 \ mod\ q.$
3. *Vrfy:* The receiver verifies a message-signature pair (m, $\sigma$) by checking if the following equality holds: $g_1^{z_1} g_2^{z_2} = ah^m$.

Note that: $g_1^{z_1} g_2^{z_2} = g_1^{mw_1+v_1}\, g_2^{mw_2+v_2} = \ g_1^{v_1} g_2^{v_2}\, (g_1^{w_1} g_2^{w_2})^m = ah^m$.

Since the above signature is a one-time signature it can only be used to sign one message with one key $(sk, vk)$. Every new message must be signed with a new key to maintain security. We propose to change the key by simply changing $(v_1, v_2)$ while keeping $(w_1, w_2)$ unchanged. Note that with the above scheme the length of the message must be $N$-bits such that $2^N < q$, where $q$ is the order of the group $G_q$. Typically $N$ is larger than 128 bits but less than 1024 bits. If a message $m$ is much larger than 1024 bits then it can be hashed into $N$-bits using a collision resistant hash function such as VSH [4-12]. We now give the proposed multicast protocol that can be used in the smart grid.

### 4.3.2. Our Protocol

The signing key (or private key of the signer) for signing the $j^{\text{th}}$ message $m_j$ is $sk = \Big((w_1, w_2), \big(v_{1j}, v_{2j}\big)\Big)$, where $(w_1, w_2)$ is such that $(h, (w_1, w_2)) \in R$, for $h = g_1^{w_1} g_2^{w_2}$, and $v_{1j}, v_{2j}$ are chosen at random from $Z_q$ for each $m_j$. The verification key (or public key) is $vk_j = (g_1^{w_1} g_2^{w_2}, g_1^{v_{1j}} g_2^{v_{2j}}) = (h, a_j)$. The verification key has to be distributed to each receiver of message $m_j$. We assume that this can be accomplished using authenticated message

transmission. The signer or sender of the multicast message can select $M$ pairs of random values from $Z_q$ , $[(v_{11}, v_{21}), (v_{12}, v_{22}), \dots, (v_{1M}, v_{2M})]$, in advance, and compute the $M$ verification keys, $vk_1, vk_2, \dots, vk_M$.[1] These keys are sent with the message. Note that $h = g_1^{w_1} g_2^{w_2}$, $g_1$, and $g_2$ are known to all the receivers via a single transmission in the beginning. The signer computes the signature of the $j^{\text{th}}$ message $m_j$ as $\sigma_j = (z_{1j}, z_{2j}) = (m_j w_1 + v_{1j} \bmod q, m_j w_2 + v_{2j} \bmod q)$ and sends a multicast message $(m_j, \sigma_j)$ to all receivers. Upon receipt of a message $(m_j, \sigma_j)$, the receiver performs verification by checking if the following equality is satisfied: $g_1^{z_{1j}} g_2^{z_{2j}} = a_j h^{m_j}$. In order to verify the message-signature pair, $(m_j, \sigma_j = (z_{1j}, z_{2j}))$, the receiver should know the values $g_1$, $g_2$, $h$, and $a_j$. All receivers receive $g_1$, $g_2$, and $h$ in the beginning. The sender computes $M$ verification keys and sends them to all receivers so the receiver knows $a_j$.

Assume that $G_q$ is the quadratic residue subgroup of $Z_p^*$, where $p$ is a strong prime ($p$ is a strong prime if $p = 2q + 1$ and $q$ is prime). Therefore if $q$ can be expressed in $N$-bits then $p$ can be expressed in at most $(N + 1)$-bits. The verification key for $M$ messages consists of $g_1$, $g_2$, $h$, and $(a_1, a_2, \dots, a_M)$ which require $(M + 3)(N + 1)$-bits. The number of verification key bits per message is $\frac{(M+3)(N+1)}{M}$, which is $N + 1$ , for $M \gg 3$.

## 4.4. Best Known One Time Signature Scheme in the Smart Grid

The best known multicast protocol is the one in [4-1]. It is adapted from one time signature scheme provided in [4-8]. In the following section, we will describe scheme from [4-1].

[1] Verification keys can also be computed each time a new message is to be transmitted.

### 4.4.1. Tunable Signing and Verification

OTS scheme in [4-1] is a combination of two extreme approaches namely Heavy Signing Light Verification (HSLV) and Light Signing Heavy Verification (LSHV). The combined scheme achieves a tradeoff between the two and is called Tunable Signing and Verification (TSV). The TSV scheme is as follows:

1. *Gen:* Generate $t$ different random *l-bit* strings $(s_1, s_2, \dots, s_t)$. For each $s_i$, generate a one-way chain of length $w+1$, i.e., $s_i \to f(s_i) \to \cdots \to f^w(s_i)$. The *t* chains form the private key $SK$. The public key is $PK = (v_1, v_2, \dots, v_t)$, where $v_i = f^{w+1}(s_i)$.
2. *Sign:* To sign a message $m$, compute $h = H(m|c)$, where $c$ is a counter with initial value 0. Call $SPLIT\ (h)$ that takes bit string $h$ as input and outputs integers $i_1, i_2, \dots, i_k$. This function splits $h$ into $k$ substrings $h_1, h_2, \dots, h_k$ of $\log_2 t$ bits each and interprets each $h_j$ as an integer $i_j$. All $i_j$ from $SPLIT\ (h)$ should be different and the $i_j$ within the same group should be sorted in the decreasing order; otherwise, increase $c$ by 1 and repeat the above process. The signature of *m* is $(c, f^{w-w_{q_1}}(s_{i_1}), \dots, f^{w-w_{q_k}}(s_{i_k}))$.
3. *Vrfy:* To verify a signature $(c', (s'_1, s'_2, \dots, s'_k))$ over message $m$, compute $h = H(m|c')$. Call $SPLIT\ (h)$. Check if 1) all $i_j$ from $SPLIT\ (h)$ are different, 2) the $i_j$ in the same group are sorted in the decreasing order and $f^{w_{q_j}+1}(s'_j) = v_{i_j}$ for each $j$.

Note that if $k$ elements of the signature are divided into $g$ groups $\mathbb{G}_1, \mathbb{G}_2, \dots, \mathbb{G}_g$ and $n_r$ $(r \in [1, g])$ denote the size of group $\mathbb{G}_r$. Then $\mathbb{G}_1$ contains $n_1$ elements, $\mathbb{G}_2$ contains $n_2$ elements etc. Let $\mathbb{G}_{q_i}$ denote the group to which the $i^{\text{th}}$ $(i \in [1, k])$ element of the signature belongs. Each element in group $\mathbb{G}_r$ is verified with $w_r + 1, (w_r \geq 0)$ one-way function invocations where, $w = max\ \{w_1, \dots, w_g\}$.

*Protocol 4.4.1.1:*

Starting from $t$ random values $s_{\langle 1,1\rangle}, s_{\langle 1,2\rangle}, \dots, s_{\langle 1,t\rangle}$, the sender generates $t$ one-way chains of length $d+1, (d \gg k)$ and stores them as series of keys. The initial public key is $PK_0 = (s_{\langle d+1,1\rangle}, s_{\langle d+1,2\rangle}, \dots, s_{\langle d+1,t\rangle})$. Assume $PK_0$ is distributed to each receiver securely, e.g., via unicast message authenticated by HMAC. The initial private key $SK_0$ consists of $t$ $k$-element chain segments that are adjacent to $PK_0$.

When a signature is generated and revealed, the key values included in the signature and those that can be generated by applying the one-way function over them are exposed. Thus, the sender refreshes its private key by replacing any exposed key values with their predecessors in the same chain. Also, a receiver updates its public key by replacing corresponding old key values with the new values from the signature. [4-1]

## 4.5. Comparison

In the following section, we compare our protocol with [4-1] with respect to computation time for signing and verification. We will also compare length of signing and verification keys, length of signature, storage cost, and pre-computation cost as shown in Table 4.1. We will provide details of the comparison in following subsections.

### 4.5.1. Computation Time

#### 4.5.1.1. Signing Algorithm

The time to compute a signature $\sigma = (z_1, z_2) = (mw_1 + v_1 \, mod \, q, mw_2 + v_2 \, mod \, q)$, in our method is the time taken for two multiplications and two additions modulo $q$. In addition to it, there might be a need for performing one hash. Hardware complexity for computing modulo multiplication is $O((\log N)^{1+\epsilon})$ operations where $\epsilon = 1$, when ordinary multiplication is

used. Here $N$ is number of bits required to represent $q$. The cost of basic VSH (1024-bit hash) algorithm is $O\left(\frac{(\log N)^{1+\epsilon}}{k}\right) = O((logN)^{\epsilon} loglogN)$operations per message bit [4-12]. The time taken to compute the signature in [4-1] is time required to compute $c$ iterations of hash functions. Where, $c = (t^k/[t(t-1)\dots(t-k+1)]) * \prod_{r=1}^{g} n_r!$.This value is of order $k!$ for $t \gg k$. In typical setting of $N = 1023$, for message of length $2046$ bits our scheme will require $1$ application of hash followed by $2 * (100)$ cycles of CPU operations for performing two multiplications. Cost of performing one hash is $332$ operations per message bit, hence total of 679272 cycles for hash are required. Thus to Sign a message of length $2kb$ our scheme will require $679472$ cycles. However, TSV will require minimum of 8! i.e. $40320$ applications of cryptographic hash functions. Standard $160$ bit cryptographic hash functions such as SHA1 as suggested in [4-8] requires at least 8.3 cycles per byte [4-13]. Hence for a $2046$-bit message TSV will require $40320 * (2046 * 8.3/8) = 85588272$ cycles. Hence, our scheme performs $125$ times better than TSV for message of length $2kb$ using basic algorithm for modular multiplication.

### 4.5.1.2. Verification Algorithm

The time to verify a signature in our method is the time to compute one hash plus the time to compute $g_1^{z_1} g_2^{z_2} = ah^m$, which consists of 2.5 exponentiations (1.5 for performing $g_1^{z_1} g_2^{z_2}$ and one for performing $h^m$) and two multiplications in group $G_q$. One exponentiation is equal to a maximum of $2N$ multiplications [4-14]. Hence we will require $5N + 2$ multiplications. Therefore total verification cost will be $O(N(logN)^2)$ and is independent of length of message $m$. Note that if the length of $m$ is greater than $N$ then it can be hashed into a string of length less than $N$. Verification time in [4-1] requires one computation of hash and between 0 to

Table 4.1. Comparison with Cao's Protocol.

| Here $\mu = t^k/[t(t-1)\dots(t-k+1)]$. | | | | | | | |
|---|---|---|---|---|---|---|---|
| Scheme | Computation Cost | | Pre Computation Cost | Key Length | | Signature Length | Storage Cost |
| | Sign | Verify | | Sign | Verify | | |
| HSLV | $k!\,\mu$ applications of hash function | k+1 one-way function invocations | $t$ applications of one-way function | $tl$ | $tl$ | $kl + \log k!\,\mu$ | $tl$ |
| LSHV | $\mu$ applications of hash function | $\frac{k(k+1)}{2} + 1$ one-way function invocations | $tk$ applications of one-way function | $kwl$ | $tl$ | $kl + \log \mu$ | $lt(w+1)$ |
| TSV | $\mu \prod_{r=1}^{g} n_r!$ applications of hash function | $1 + k + \sum_{r=1}^{g} n_r w_r$ One-way function invocations | $t(d+1)$ applications of a one-way functions for signature of $2td/(k(k+1))$ messages | $kdl$ | $tl$ | $kl + \log[\mu \prod_{r=1}^{g}(n_r!)]$ | $lt(d+1)$ for signature of $2td/(k(k+1))$ messages |
| Our Scheme | 1 application of hash function + 2 modular multiplications | 2.5 exponentiations or one-way function invocations | 3 exponentiations or applications of one-way functions | $4N$ | $4(N+1)$ | $2N$ | Sender: 2N +2MN<br>Receiver : 3(N+1) |

$k(k-1)/2$ computations of one-way function $f(.)$. If we assume the computational complexity of one-way function to be the same as that of exponentiation because large modular exponentiations in itself are one-way functions, then verification cost of our scheme is better than TSV as only 2.5 applications of one-way functions are required in our case. Also, in TSV an attempt to minimize verification cost increases signing cost. Also, to select the optimum value of the number of one-way functions, $C$, required for verification, offline computation of $O(k^3)$ is required. We do not need any offline computation.

### 4.5.2. Key Generation and Public Key Distribution Cost

TSV method requires the pre-computation of $t$ one-way chains of length $(d+1)$. To accomplish this requires $t(d+1)$ applications of a one-way function. This pre-computation allows the computation of the signature for $2td/(k(k+1))$ messages. Assuming that the chains

are depleted as they are revealed in the signatures, an additional communication cost is caused by transmission of the public key, which is on average $\frac{k(k+1)}{2d} l$-bits per message. However, our method incurs fixed cost for selection of generators $g_1$ and $g_2$ as explained earlier. Also, sampling of $(w_1, w_2)$ such that $(h, (w_1, w_2)) \in \mathcal{R}$, where $h = {g_1}^{w_1} {g_2}^{w_2}$. In addition to this to sign M different messages it requires selection of 2M random values, $[(v_{11}, v_{21}), (v_{12}, v_{22}), \dots, (v_{1n}, v_{2M})]$, and the computation of $a_j = g_1^{v_{1j}} g_2^{v_{2j}}$, for j = 1, …, M. This requires 1.5M exponentiations and M multiplications in group $G_q$. As explained earlier the size of $a_j$ is N+1 bits and hence that will be the cost of public key distribution per message. For a typical setting in TSV, where $t = 1024$, $d = 1024$, $k = 8$, and $l = 80$ i.e. when message is of size 80kb, $(2 * 1024 * 1024)/(8 * 9) = 29127$ messages can be authenticated with $1024$ chains i.e. $1024 * 1025 = 1049600$ applications of a one-way functions. Also key distribution cost is $0.4B$ per message. Hence, it will be 11650.8B for 29127 messages. In our case to authenticate $29127$ messages of size $80kb$, we need to perform $(1.5 + 1.5 * 29127) = 43692$ applications of one-way functions or that many exponentiations. Distribution cost of such key is $1024$ bits per message. This will be $1.25\%$ of message size for the current example and is hence negligible. Therefore, our scheme performs 24 times better than TSV in terms of key generation. We suggest computing a new public key every time a new message is to be transmitted. In that case, 1.5 extra exponentiations performed online will not add much computational overhead for sender as compared to TSV. We can say that an addition of 1.5 one-way functions and increase of $1kb$ in communication overhead will not make much of a difference for large messages.

### 4.5.3. Key Length

The length of the verification key in TSV is $tl$-bits while the length of the verification key in our method is the length of four group elements, $h, a_j, g_1$, and $g_2$, which is $4(N+1)$-bits. The length of the signing key in TSV is $kdl$-bits where k < t. The length of the signing key, $sk = \left((w_1, w_2), (v_{1j}, v_{2j})\right)$, in our method is $4N$-bits. For example, when $t = 1024, l = 80, d = 1024,$ and $N = 1023$, the verification key length in our scheme is 4096-bits, out of which 3072-bits are transmitted in the beginning itself. However, verification key length in TSV will be 81920-bits, which is transmitted in the beginning. Our scheme achieves improvement in the verification key length that is transmitted in the beginning and is to be stored at the receiver by a factor of 26. This is useful in applications like demand response, operation and control, and in-substation protection where receiver storage is stringent. For the same settings, signing key length in our scheme is 4092-bits. But sender has to store $2MN$-bits to authenticate $M$ messages but as we will discuss in sub section on storage cost this can be avoided. Signing key length in TSV is $655kb$. Hence we can say our scheme performs better than TSV in terms of key length.

### 4.5.4. Signature Length

The signature length in TSV is approximately $kl$ bits. The length of the signature $\sigma = (z_1, z_2) = (mw_1 + v_1 \, mod \, q, mw_2 + v_2 \, mod \, q)$, in our method is $2N$ bits. For typical settings of $l = 80, k = 8, and \, N = 1023$: signature length for our scheme will be $2kb$, while that for TSV will be 640-bits. Hence our scheme will add more communication overhead than TSV. However 2kb is not much of overhead. It may cause problems only in the settings where bandwidth requirements are stringent. However, using lower value of modulus $N$ might solve the problem in that case.

### 4.5.5. Storage Cost

If the output of the one-way function is $l$-bits then the total number of bits that need to be stored at the signer is $lt(d+1)$ bits. In our method the signer must pre-compute and store $(v_{11}, v_{21}, v_{12}, v_{22}, \dots, v_{1M}, v_{2M})$. If each $v_{1j}, v_{2j}$ requires $2N$-bits, then the storage requirement $(2MN + 2N)$-bits. Receiver has to store $g_1, g_2, h$. This will require $3(N+1)$ bits. Hence for our particular case of $N = 1023, t = 1024, l = 80, k = 8, d = 1024$: Storage cost at receiver is around $80Mb$. While for us its mere $4kb$. We can reduce the storage cost of the sender by using a pseudorandom number generator to output $(v_{1j}, v_{2j})$ using $j$ as seed. This will still add $N$ cycles to signing cost which is very small compared to 68k cycles for a 2kb message. Table 4.2 gives a particular example for the comparison of our protocol with that of [4-1].

Table 4.2. Particular Case of $\mathrm{l} = 80, \mathrm{t} = 1024, \mathrm{k} = 8, \mathrm{d} = 1024$, and $\mathrm{N} = 1024$.

| Scheme | Approximate Computation Cost | | Pre Computation Cost | Key Length | | Signature Length | Storage Cost |
|---|---|---|---|---|---|---|---|
| | Sign | Verify | | Sign | Verify | | |
| HSLV | 3.2G-CPU operations | 9 invocations of one-way functions | 1024 invocations of one-way functions | 80kb | 80kb | 0.640kb | 80kb |
| LSHV | 85k-CPU operations | 37 invocations of one-way functions | 8k invocations of one-way functions | 655kb | 80kb | 0.640kb | 80Mb |
| TSV | 3.2G-CPU operations | Minimum 9 invocations of one-way functions | 1M invocations of one-way functions for 29k messages of size 80kb | 655kb | 80kb | 0.640kb | 80Mb |
| Our Scheme | 27M-CPU operations | 3 invocations of one-way functions | 43k invocations of one-way functions for 29k messages of size 80kb | 4kb | 4kb | 2kb | Sender: 59Mb<br><br>Receiver: 3kb |

## 4.6. Conclusion

Multicast authentication while being a key-enabler for several critical control, command and monitoring data streams, poses unique challenges in smart grids. These challenges stem from the need to balance security with competing objectives of communication bandwidth, storage and computation overheads. This paper proposes for the first time, the use of one-time signatures from sigma protocols for multicast authentication in the smart grid. Sigma protocols are fundamentally appealing in authentication systems where one entity needs to establish the veracity of a statement to another entity without explicitly revealing its contents. Here, we construct an OTS scheme using the Okamoto protocol (a 3 move protocol with the witness hiding property) for multicast authentication. The proposed method outperforms the currently best known approach for multicast authentication [4-5] by: reducing the burden on sender and receiver computations (2-3 orders), pre-computation cost (2 orders), and storage overhead (2 orders) at a very modest increase (four fold) in signature size. With the current methods for multicast authentication in smart grids, a high level of security is obtained at the expense of increases in storage, computation and bandwidth. The proposed authentication scheme overcomes this fundamental limitation and enables the efficient use of resource constrained field devices thus paving the way for the implementation of sigma protocols in the smart grid environment.

## 4.7. References

[4-1] Li, Qinghua, and Guohong Cao. "Multicast authentication in the smart grid with one-time signature." *Smart Grid, IEEE Transactions on* 2, no. 4 (2011): 686-696.

[4-2] Mohsenian-Rad, A-H., and Alberto Leon-Garcia. "Distributed internet-based load altering attacks against smart power grids." *Smart Grid, IEEE Transactions on* 2, no. 4 (2011): 667-674.

[4-3] Madani, Vahid, Damir Novosel, and Roger King. "Technological breakthroughs in system integrity protection schemes'." In *Power System Computation Conference (PSCC)*. 2008.

[4-4] "Guidelines for Smart Grid Cyber Security: Vol. 2, Privacy and the Smart Grid", *The Smart Grid Interoperability Panel- Cyber Security Working Group*, August 2010.

[4-5] Lamport, Leslie. *Constructing digital signatures from a one-way function*. Vol. 238. Technical Report CSL-98, SRI International, 1979.

[4-6] Rabin, Michael O. "Digitalized signatures." *Foundations of secure computation*78 (1978): 155-166.

[4-7] Perrig, Adrian. "The BiBa one-time signature and broadcast authentication protocol." In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pp. 28-37. ACM, 2001.

[4-8] Reyzin, Leonid, and Natan Reyzin. "Better than BiBa: Short one-time signatures with fast signing and verifying." In *Information Security and Privacy*, pp. 144-153. Springer Berlin Heidelberg, 2002.

[4-9] Goldreich, Oded. "Foundations of Cryptography, volume I." *Basic Tools*, 2003.

[4-10] Hazay, Carmit, and Yehuda Lindell. Efficient secure two-party protocols: Techniques and constructions. Springer, 2010.

[4-11] Damgard, Ivan, *Lecture Notes on Cryptographic Protocol Theory*. Department of Computer Science, AARHUS University, 2011.

[4-12] Contini, Scott, Arjen K. Lenstra, and Ron Steinfeld. "VSH, an efficient and provable collision-resistant hash function." In *Advances in Cryptology-EUROCRYPT 2006*, pp. 165-182. Springer Berlin Heidelberg, 2006.

[4-13] Nakajima, Junko, and Mitsuru Matsui. "Performance analysis and parallel implementation of dedicated hash functions." In *Advances in Cryptology—EUROCRYPT 2002*, pp. 165-180. Springer Berlin Heidelberg, 2002.

[4-14] Stinson, Douglas Robert. *Cryptography: theory and practice*. CRC press, 2002.

[4-15] Pieprzyk, Josef, Huaxiong Wang, and Chaoping Xing. "Multiple-time signature schemes against adaptive chosen message attacks." In *Selected Areas in Cryptography*, pp. 88-100. Springer Berlin Heidelberg, 2004.

[4-16] Neumann, William D. "HORSE: an extension of an r-time signature scheme with fast signing and verification." In *Information Technology: Coding and Computing, 2004. Proceedings. ITCC 2004. International Conference on*, vol. 1, pp. 129-134. IEEE, 2004.

[4-17] Wang, Qiyan, Himanshu Khurana, Ying Huang, and Klara Nahrstedt. "Time valid one-time signature for time-critical multicast data authentication." In*INFOCOM 2009, IEEE*, pp. 1233-1241. IEEE, 2009.

[4-18] Chang, Shang-Ming, Shiuhpyng Shieh, Warren W. Lin, and Chih-Ming Hsieh. "An efficient broadcast authentication scheme in wireless sensor networks." In*Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pp. 311-320. ACM, 2006.

[4-19] Berbecaru, Diana. "Performance of two one-time signature schemes in space/time constrained environments." In *Wireless Pervasive Computing (ISWPC), 2010 5th IEEE International Symposium on*, pp. 238-243. IEEE, 2010.

[4-20] Bicakci, Kemal. "Pushing the limits of one-time signatures." In *Proceedings of the 2nd international conference on Security of information and networks*, pp. 249-257. ACM, 2009.

# CHAPTER 5. CONCLUSIONS

Sensor networks have become part of our day to day life and hence are hot topic for research these days. Smart-grid is an evolving sensor network technology. They are found useful in many applications related to power grid. Power grids exchange many sensitive messages e.g. price/kW, energy need, company offers, control signals etc. These messages are sent over either wired or wireless network connecting these power sensor nodes and base station. Hence there is need to protect this information. In this thesis we concentrate on protecting integrity of different messages in smart grid networks using new Message Authentication Codes (MAC).

The smart gird networks have different requirements than regular data networks and hence they need to be treated separately. They exchange heavy messages but have limited resources. Hence, we present a MAC that performs faster authentication of long messages in very short time consuming few hardware resources. In this thesis we presented three different papers targeted towards providing fast authentication for long messages in short time which have been recently accepted or submitted for publication as three different papers.

The first paper proposed that Multiple Input Shift Register (MISR) is $\epsilon$-*universal* hash function. We proved that it is faster as well as hardware efficient than existing universal hash functions and hence MISR can be used in fast authentication of long messages in resource constrained environment. Software implementation results of this scheme indicate that the time required to sign or verify a message is reduced by two order compared to existing schemes. The second paper proposes a variable length message authentication code (MAC) based on MISR for authenticated communication between metering nodes and collection nodes in smart grid networks. In this paper we proved that the new variable length MAC scheme is theoretically

secure. The MAC has two order better time performance compared to existing hash based MAC, HMAC.

The third paper focuses on multicast authentication in smart grid networks. We propose a novel approach of using one-time signatures (OTS) from sigma protocols for multicast authentication in smart grid. When compared with the currently best known OTS scheme Tunable Signing and Verification, the proposed scheme yields a three order improvement in time performance at a very modest increase (four fold) in signature size.

Throughout this thesis we used different security definitions of authentication schemes in order to prove the security of a scheme. While in some real-world applications, any scheme is considered to be safe if no effective attack is known for it; provable security can definitely help to guarantee a scheme's security as long as the underlying hard problem is not solved.

Therefore, in this thesis we provide provably secure message authentication schemes useful in smart grid and other resource constrained sensor networks. The first two schemes can be used for faster authentication of heavy messages in resource constrained sensor networks. The schemes are proved to be faster than existing schemes in terms of signing and verification cost, hardware requirement, and communication overhead.

# REFERENCES

This thesis contains three papers (chapters 2 to 4). The references of each paper appear at the end of the chapter covering that paper. Other references are listed below.


[1-1] Katz, Jonathan, and Yehuda Lindell. *Introduction to modern cryptography*. Chapman & Hall, 2008.

[1-2] Carter, J. Lawrence, and Mark N. Wegman. "Universal classes of hash functions." *Journal of computer and system sciences* 18, no. 2 (1979): 143-154.

[1-3] Black, John, Shai Halevi, Hugo Krawczyk, Ted Krovetz, and Phillip Rogaway. "UMAC: Fast and secure message authentication." In *Advances in Cryptology—CRYPTO'99*, pp. 216-233. Springer Berlin Heidelberg, 1999.

[1-4] Krawczyk, Hugo. "LFSR-based hashing and authentication." In *Advances in Cryptology—CRYPTO'94*, pp. 129-139. Springer Berlin Heidelberg, 1994.

[1-5] Vosoughi, Aida, and Rajendra Katti. "Fast Message Authentication Code for Multiple Messages with Provable Security." In *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pp. 1-5. IEEE, 2010.

[1-6] Hamlyn, Alexander, Helen Cheung, Todd Mander, Lin Wang, Cungang Yang, and R. Cheung. "Network security management and authentication of actions for smart grids operations." In *Electrical Power Conference, 2007. EPC 2007. IEEE Canada*, pp. 31-36. IEEE, 2007.

[1-7] Ericsson, Göran N. "Cyber security and power system communication—essential parts of a smart grid infrastructure." *Power Delivery, IEEE Transactions on* 25, no. 3 (2010): 1501-1507.

[1-8] Metke, Anthony R., and Randy L. Ekl. "Smart grid security technology." In*Innovative Smart Grid Technologies (ISGT), 2010*, pp. 1-7. IEEE, 2010.

[1-9] Fouda, Mostafa M., Zubair Md Fadlullah, Nei Kato, Rongxing Lu, and Xuemin Shen. "A lightweight message authentication scheme for smart grid communications." *Smart Grid, IEEE Transactions on* 2, no. 4 (2011): 675-685.

[1-10] Lamport, Leslie. *Constructing digital signatures from a one-way function*. Vol. 238. Technical Report CSL-98, SRI International, 1979.

[1-11] Rabin, Michael O. "Digitalized signatures." *Foundations of secure computation*78 (1978): 155-166.

[1-12] Perrig, Adrian. "The BiBa one-time signature and broadcast authentication protocol." In *Proceedings of the 8th ACM conference on Computer and Communications Security*, pp. 28-37. ACM, 2001.

[1-13] Reyzin, Leonid, and Natan Reyzin. "Better than BiBa: Short one-time signatures with fast signing and verifying." In *Information Security and Privacy*, pp. 144-153. Springer Berlin Heidelberg, 2002.

[1-14] Li, Qinghua, and Guohong Cao. "Multicast authentication in the smart grid with one-time signature." *Smart Grid, IEEE Transactions on* 2, no. 4 (2011): 686-696.

# APPENDIX. MHMAC SOURCE CODE

*Main_File.cpp*

```
# include <MHMAC.h>
# include <conio.h>

int main()
{
	string MessageToSign;
	cout<<"Enter Text :"<<endl;
	cin >> MessageToSign;
	MHMAC MAC;
	MAC.MHMAC_Gen();
	string MESSAGE=MessageToSign;
	string MHMAC_Sign= MAC.MHMAC_Sign(MESSAGE);
	bool b=MAC.MHMAC_Vrfy(MESSAGE,MHMAC_Sign);
	cout<<"Verify :"<<b<<endl;
	_getch();
	return 1;
}
```

*MHMAC.h*

```
# ifndef _MHMAC
# define _MHMAC

# include <MISRH.h>
# include <DHKeyXchnge.h>


class MHMAC
{
	byte AESkey[AES::DEFAULT_KEYLENGTH];
	byte AESiv[AES::BLOCKSIZE];
	uint64_t gx_high;
	uint64_t gx_low;
	string Msg;
	string Signature;
	string counter;
public:

	void MHMAC_Gen();
	string MHMAC_Sign(string);
	bool MHMAC_Vrfy(string,string);
};
# endif
```

*MHMAC.cpp*

```
# include <MHMAC.h>
# include <CounterGen.h>
```

```
void MHMAC::MHMAC_Gen()
{
	cout<<"Generating Keys"<<endl;
	DHKeyXchnge DH_1;
	//SecByteBlock & RefAESKey = AESkey(RIPEMD128::DIGESTSIZE);
	static const int size = RIPEMD128::DIGESTSIZE;
	//byte AESKEY[16];
	//AESkey = AESKEY;
	DH_1.ExchangeKey();
	for (int i=0;i<AES::DEFAULT_KEYLENGTH;i++)
	{
		AESkey[i]=DH_1.key[i];
	}
	for (int i=0;i<AES::BLOCKSIZE;i++)
	{
		AESiv[i]=DH_1.iv[i];
	}
	// Pretty print key
	string encodedkey,encodediv;
	encodedkey.clear();
	StringSource(AESkey, sizeof(AESkey), true,
		new HexEncoder(
			new StringSink(encodedkey)
		) // HexEncoder
	); // StringSource
	cout << "key: " << encodedkey << endl;
	encodediv.clear();
	StringSource(AESiv, sizeof(AESiv), true,
		new HexEncoder(
			new StringSink(encodediv)
		) // HexEncoder
	); // StringSource
	cout << "iv: " << encodediv << endl;

}

string MHMAC::MHMAC_Sign(string Msg)
{
			cout<<"Signing..."<<endl;
			CounterGen r;
			counter = r.getCounterValue();
			cout<<"Counter :"<<counter<<endl;

			STR2H conv1(Msg);
			string MsgHex = conv1.HString();
			int NoHexChars = conv1.Get_L();
			int NofU64 = ceil(double(NoHexChars/16));
			MessageArray MsgtoMISRH(MsgHex);
			MsgtoMISRH.setMsgLen();
			uint64_t *MsgtoMISRHPtr = MsgtoMISRH.GetArray();
			int MsgtoMISRHLen = MsgtoMISRH.getMsgLen();
			MISRH MyMISR;
			MyMISR.ComputeMISRH(MsgtoMISRHPtr,MsgtoMISRHLen);
			string plainTextS = MyMISR.getRx();
			cout<<"MISRH:"<<plainTextS<<endl;
			string encodedS;
			string encodedSign;
			try
```

```
		{

		CBC_Mode< AES >::Encryption e;
		e.SetKeyWithIV(AESkey, sizeof(AESkey), AESiv);

		// The StreamTransformationFilter removes
		//  padding as required.
		string Sign;
		StringSource s(counter, true,
			new StreamTransformationFilter(e,
				new StringSink(Sign)
			) // StreamTransformationFilter
		); // StringSource
		// Pretty print
		encodedS.clear();
		StringSource(Sign, true,
		new HexEncoder(
			new StringSink(encodedS)
			) // HexEncoder
		); // StringSource


		for(int i = 0 ; i<plainTextS.size(); i++)
			plainTextS[i] ^= encodedS[i];

		Signature = plainTextS;
		encodedSign.clear();
		StringSource(Signature, true,
		new HexEncoder(
			new StringSink(encodedSign)
			) // HexEncoder
		); // StringSource

#if 0
		StreamTransformationFilter filter(e);
		filter.Put((const byte*)plainTextS.data(), plainTextS.size());
		filter.MessageEnd();

		const size_t ret = filter.MaxRetrievable();
		Signature.resize(ret);
		filter.Get((byte*)Signature.data(), Signature.size());
#endif
	}
	catch(const CryptoPP::Exception& e)
	{
		cerr << e.what() << endl;
		exit(1);
	}

	/*********************************\
	\*********************************/


	cout << "MHMAC: " << encodedSign << endl;

	return encodedSign;
}

bool MHMAC::MHMAC_Vrfy(string MsgtoVrfy, string MAC)
```

```
{
		cout<<"Verifying..."<<endl;
		STR2H conv2(MsgtoVrfy);
		string MsgHexV = conv2.HString();
		int NoHexCharsV = conv2.Get_L();
		int NofU64V = ceil(double(NoHexCharsV/16));
		MessageArray MsgtoMISRHV(MsgHexV);
		MsgtoMISRHV.setMsgLen();
		uint64_t *MsgtoMISRHPtrV = MsgtoMISRHV.GetArray();
		int MsgtoMISRHLenV = MsgtoMISRHV.getMsgLen();
		MISRH MyMISR;
		MyMISR.ComputeMISRH(MsgtoMISRHPtrV,MsgtoMISRHLenV);
		string plainTextV = MyMISR.getRx();
		cout<<"MISRH:"<<plainTextV<<endl;
		string encodedV,MACV,encodedVrfy;
		try
		{

		CBC_Mode< AES >::Encryption eV;
		eV.SetKeyWithIV(AESkey, sizeof(AESkey), AESiv);

		// The StreamTransformationFilter removes
		//  padding as required.
		string Vrfy;
		StringSource s(counter, true,
			new StreamTransformationFilter(eV,
				new StringSink(Vrfy)
			) // StreamTransformationFilter
		); // StringSource
		// Pretty print
		encodedV.clear();
		StringSource(Vrfy, true,
		new HexEncoder(
			new StringSink(encodedV)
			) // HexEncoder
		); // StringSource


		for(int i = 0 ; i<plainTextV.size(); i++)
			plainTextV[i] ^= encodedV[i];

		MACV = plainTextV;



#if 0
		StreamTransformationFilter filter(e);
		filter.Put((const byte*)plainTextV.data(), plainTextV.size());
		filter.MessageEnd();

		const size_t ret = filter.MaxRetrievable();
		MACV.resize(ret);
		filter.Get((byte*)MACV.data(), MACV.size());
#endif
	}
	catch(const CryptoPP::Exception& e)
	{
		cerr << e.what() << endl;
		exit(1);
```

```
	}

	/*********************************\
	\*********************************/

	// Pretty print
	encodedVrfy.clear();
	StringSource(MACV, true,
		new HexEncoder(
			new StringSink(encodedVrfy)
		) // HexEncoder
	); // StringSource
	cout << "MHMAC: " << encodedVrfy << endl;
	return (encodedVrfy==MAC);
}
```

*MISRH.h*

```
# ifndef _MISRH
# define _MISRH
# include <MessageArray.h>

class MISRH
{
	uint64_t Gxh;
	uint64_t Gxl;
	uint64_t Rxh,Rxl;
	string Rx;
public:
MISRH();
void ComputeMISRH(uint64_t *MsgPtr, int MsgSize);
string getRx();
};
#endif
```

*MISRH.cpp*

```
# include <MISRH.h>
string tostring(uint64_t val)
{
  std::ostringstream o;
	o << hex<< val;
	return o.str();
}

MISRH::MISRH()
{
	Gxh = 0x8000000000000000u;
	Gxl = 0x0000000000000043u;
	Rxh=0u;
	Rxl=0u;
}
```

```
void MISRH::ComputeMISRH(uint64_t *MsgPtr, int msgsize)
{
	uint64_t temph,temp1h,temp2h,templ,temp1l,temp2l;
	for (int i=0;i<msgsize;i+=2)
	{
		// taps: 128 07 02 1; characteristic polynomial: x^128 + x^7 + x^2 + x + 1

		temph = Rxh;
		temp1h=temph >> 1;
		//temp2h=temph & 1u;
		templ = Rxl ;
		if ((temph & 1u) == 1)
		{
			temp1l = (templ >> 1) | (0x8000000000000000u);
		}
		else
		{
			temp1l = (templ >> 1);
		}
		//temp2l=templ & 1u;
		if ((templ & 1u)==1u)
		{
			Rxh = (temp1h) ^ ((0xFFFFFFFFFFFFFFFFu) & (Gxh)) ^ (*MsgPtr) ;
			Rxl = (temp1l) ^ ((0xFFFFFFFFFFFFFFFFu) & (Gxl)) ^ (*(MsgPtr++)) ;
		}
		else
		{
			Rxh = (temp1h) ^ ((0x0u) & (Gxh)) ^ (*MsgPtr) ;
			Rxl = (temp1l) ^ ((0x0u) & (Gxl)) ^ (*(MsgPtr++)) ;
		}

		/*cout<<hex<<Rxh<<endl;
		cout<<hex<<Rxl<<endl;*/

	}
	/*cout<<"Final Rx:"<<endl;
	cout<<hex<<Rxh<<endl;
	cout<<hex<<Rxl<<endl;*/

}

string MISRH::getRx()
{
	string Hashh=tostring(Rxh);
	string Hashl=tostring(Rxl);
	Rx = Hashh.append(Hashl);
	return Rx;
}
```

*CounterGen.h*

```
# ifndef _COUNTERGEN
# define _COUNTERGEN

# include <iostream>
# include <string>
# include <sstream>
```

```
using namespace std;
# include <stdint.h>

class CounterGen
{
	static uint64_t counter_low;
	static uint64_t counter_high;

public:
	static string getCounterValue();
};


# endif
```

*CounterGen.cpp*

```
# include <CounterGen.h>

uint64_t CounterGen::counter_high = rand();
uint64_t CounterGen::counter_low = rand();

string CounterGen::getCounterValue()
{
	if (counter_low == UINT64_MAX)
	{
		if (counter_high == UINT64_MAX)
		{
			counter_low = 0;
			counter_high = 0;
		}
		else
		{
			counter_high++;
		}
	}
	else
	{
		counter_low++;
	}
	stringstream ss;
	ss << counter_high << counter_low;
	string str = ss.str();
	return str;
}
```

*MessageArray.h*

```
#ifndef _MessageArray
#define _MessageArray

# include <string>
```

```
# include <memory>
# include <math.h>
# include <Str2H.h>
# include <H2Uint64.h>

class MessageArray
{
	string MsgArray;
	uint64_t *MsgPtr;
	int MsgLen;
public:
	MessageArray(){MsgArray = "";}
	MessageArray(string msg){MsgArray = msg;}
	uint64_t * GetArray();
	void setMsgLen(){MsgLen=ceil(double(MsgArray.length()/16));}
	int getMsgLen(){return MsgLen;}
	void DispArray();
	~ MessageArray();
};

#endif
```

*MessageArray.cpp*

```
#include <MessageArray.h>

uint64_t * MessageArray::GetArray()
{
	H2UINT64 MessageNo;

	int k=0;
	try
	{
		MsgPtr = new uint64_t[MsgLen];
		uint64_t *uint64_ptr = MsgPtr;

		for (int j=0; j<MsgLen; j++)
		{
			*uint64_ptr = MessageNo.convert(MsgArray.substr(k,16));
			k+=16;
			uint64_ptr++;
		}
		return MsgPtr;
	}
	catch (bad_alloc ba)
	{
		cout<<"Bad Alloc"<<endl;
	}

}

void MessageArray::DispArray()
{
	uint64_t *dispPtr = MsgPtr;
	for (int j=0; j<MsgLen; j++)
	{
	cout << *dispPtr <<endl;
	dispPtr++;
```

```
		}
}

MessageArray::~MessageArray()
{
		MessageArray::MsgArray = "";
}
```

*H2Uint64.h*

```
/*
H2Uin64.h
To conver Hex representation of String to
of it
*/

#ifndef H2UINT64_H
#define H2UINT64_H

# include <iostream>
# include <string>
using namespace std;
# include <stdint.h>


class H2UINT64
{
		string SH;
		uint64_t Uint64_No;
public:
		H2UINT64(){SH="";}
		H2UINT64(string S){SH=S;}
		uint64_t convert(std::string &s);
};

#endif
```

*H2Uint64.cpp*

```
# include <H2Uint64.h>


uint64_t H2UINT64::convert(std::string &s)
{
   std::string::iterator i;
   std::string digits = "0123456789abcdefABCDEF";
   uint64_t result = 0;
   size_t pos = 0;
		SH = s;
   i = s.begin();

   while (i != s.end())
   {
      // search for character in hex digits set
```

```
        pos = digits.find(*i);

        // if found in valid hex digits
        if (pos != std::string::npos)
        {
            // handle upper/lower case hex digit
            if (pos > 0xf)
            {
                pos -= 6;
            }

            // shift a nibble in
            result <<= 4;
            result |= pos;
        }

        ++i;
    }

    return result;

}
```

*Str2H.h*

```
/*
Str2H.h
to convert regular string to hex representation
of it
*/

#ifndef STR2H_H
#define STR2H_H

# include <iostream>
# include <string>
# include <sstream>
# include <iomanip>
# include <iterator>

using namespace std;

class STR2H
{
        string Sin;
        string Sout;
        int Sout_L;

public:
        STR2H(){Sin=Sout="";}
        STR2H(string S){ Sin=S;}
        void getString();
        string HString();
        int Get_L(){return Sout_L;}
};
```

```
#endif
```

*Str2H.cpp*

```
# include <Str2H.h>
void STR2H::getString()
{
	cout<<"Enter String with no spaces:"<<endl;
	cin>>Sin;

}
string STR2H::HString()
{
	ostringstream result;
	result << setw(2) << std::setfill('0') << std::hex << std::uppercase;
   std::copy(Sin.begin(), Sin.end(), std::ostream_iterator<unsigned int>(result));
	Sout=result.str();
	Sout_L=Sout.length();

	int msgsize = 0;
	if (Sout_L%32 == 0)
	{
		Sout=Sout;
	}
	else
	{
		int m=32-(Sout_L%32);
		Sout=Sout;

		for (int l=0;l<m;l++)
		{
			Sout+="0";
		}
	}
	Sout_L=Sout.length();
	return Sout;
}
```

*DHKeyXchnge.h*

```
# ifndef _DHKEYXCHNGE
# define _DHKEYXCHNGE

# include <CryptIncludes.h>
# include <stdexcept>
using std::runtime_error;

#include <sstream>
using std::istringstream;
using std::stringstream;

# include <iostream>
using namespace std;
```

```
class DHKeyXchnge
{

public:
        byte iv[AES::BLOCKSIZE];
        byte key[AES::DEFAULT_KEYLENGTH];
        void ExchangeKey()
        {
                AutoSeededRandomPool rnd;
        unsigned int bits = 128;
        try
        {
                DH dh;
                dh.AccessGroupParameters().GenerateRandomWithKeySize(rnd, bits);

                if(!dh.GetGroupParameters().ValidateGroup(rnd, 3))
                        throw runtime_error("Failed to validate prime and generator");

                size_t count = 0;

                const Integer& p = dh.GetGroupParameters().GetModulus();
                count = p.BitCount();
                cout << "P (" << std::dec << count << "): " << std::hex << p << endl;

                const Integer& q = dh.GetGroupParameters().GetSubgroupOrder();
                count = q.BitCount();
                cout << "Q (" << std::dec << count << "): " << std::hex << q << endl;

                const Integer& g = dh.GetGroupParameters().GetGenerator();
                count = g.BitCount();
                cout << "G (" << std::dec << count << "): " << std::dec << g << endl;

                // http://groups.google.com/group/sci.crypt/browse_thread/thread/7dc7eeb04a09f0ce
                Integer v = ModularExponentiation(g, q, p);
                if(v != Integer::One())
                        throw runtime_error("Failed to verify order of the subgroup");
                //Generate PubPriv Pair

                //Simple DH key Generation

                SecByteBlock privKeyA(dh.PrivateKeyLength());
                SecByteBlock pubKeyA(dh.PublicKeyLength());
                dh.GenerateKeyPair(rnd, privKeyA, pubKeyA);

                SecByteBlock privKeyB(dh.PrivateKeyLength());
                SecByteBlock pubKeyB(dh.PublicKeyLength());
                dh.GenerateKeyPair(rnd, privKeyB, pubKeyB);

                Integer a, b;

                a.Decode(pubKeyA.BytePtr(), pubKeyA.SizeInBytes());
                cout << "Shared secret (A): " << std::hex << a << endl;

                b.Decode(pubKeyB.BytePtr(), pubKeyB.SizeInBytes());
                cout << "Shared secret (B): " << std::hex << b << endl;

                //Unified DH key generation
```

```
		DH2 dhA(dh), dhB(dh);

		SecByteBlock sprivA(dhA.StaticPrivateKeyLength()), spubA(dhA.StaticPublicKeyLength());
		SecByteBlock eprivA(dhA.EphemeralPrivateKeyLength()),
epubA(dhA.EphemeralPublicKeyLength());

		SecByteBlock sprivB(dhB.StaticPrivateKeyLength()), spubB(dhB.StaticPublicKeyLength());
		SecByteBlock eprivB(dhB.EphemeralPrivateKeyLength()), epubB(dhB.EphemeralPublicKeyLength());

		dhA.GenerateStaticKeyPair(rnd, sprivA, spubA);
		dhA.GenerateEphemeralKeyPair(rnd, eprivA, epubA);

		dhB.GenerateStaticKeyPair(rnd, sprivB, spubB);
		dhB.GenerateEphemeralKeyPair(rnd, eprivB, epubB);

		//Key Establishment

		if(dhA.AgreedValueLength() != dhB.AgreedValueLength())
		throw runtime_error("Shared secret size mismatch");

		SecByteBlock sharedA(dhA.AgreedValueLength()), sharedB(dhB.AgreedValueLength());

		if(!dhA.Agree(sharedA, sprivA, eprivA, spubB, epubB))
		throw runtime_error("Failed to reach shared secret (A)");

		if(!dhB.Agree(sharedB, sprivB, eprivB, spubA, epubA))
		throw runtime_error("Failed to reach shared secret (B)");

		count = std::min(dhA.AgreedValueLength(), dhB.AgreedValueLength());
		if(!count || 0 != memcmp(sharedA.BytePtr(), sharedB.BytePtr(), count))
		throw runtime_error("Failed to reach shared secret");


		Integer y, z;

		y.Decode(sharedA.BytePtr(), sharedA.SizeInBytes());
			cout << "Shared secret (A): " << std::hex << y << endl;

		z.Decode(sharedB.BytePtr(), sharedB.SizeInBytes());
			cout << "Shared secret (B): " << std::hex << z << endl;

		//Generate AES

		int aesKeyLength = RIPEMD128::DIGESTSIZE; // 16 bytes = 128 bit key
		int defBlockSize = AES::BLOCKSIZE;

	// Calculate a RIPEMD128 hash over the Diffie-Hellman session key
		//SecByteBlock key(RIPEMD128::DIGESTSIZE);
		static const int size = RIPEMD128::DIGESTSIZE;
		RIPEMD128().CalculateDigest(key, sharedA, sharedA.size());

	// Generate a random IV

		rnd.GenerateBlock(iv, AES::BLOCKSIZE);

		}
		catch(const CryptoPP::Exception& e)
	{
		cerr << e.what() << endl;
```

```
        }

        }

};
# endif
```

*CryptIncludes.h*

```
# ifndef _CRYPTINC
# define _CRYPTINC

// Includes
/***Crptographic Includes***/
# include "osrng.h"
using CryptoPP::AutoSeededRandomPool;

#include "integer.h"
using CryptoPP::Integer;

#include "secblock.h"
using CryptoPP::SecByteBlock;

#include <ripemd.h>
using CryptoPP::RIPEMD128;

#include "filters.h"
using CryptoPP::StringSink;
using CryptoPP::StringSource;
using CryptoPP::StreamTransformationFilter;
using CryptoPP::HashFilter;
using CryptoPP::HashVerificationFilter;

#include "dh.h"
using CryptoPP::DH;

#include "secblock.h"
using CryptoPP::SecByteBlock;

#include <dh2.h>
using CryptoPP::DH2;

#include "nbtheory.h"
using CryptoPP::ModularExponentiation;

#include "cryptlib.h"
using CryptoPP::Exception;

#include "aes.h"
using CryptoPP::AES;
using CryptoPP::AESEncryption;
using CryptoPP::AESDecryption;

#include "ccm.h"
using CryptoPP::CFB_Mode;
using CryptoPP::CBC_Mode;
```

```
# include <base64.h>
using CryptoPP::Base64Encoder;

# include <hex.h>
using CryptoPP::HexEncoder;

# endif
```