Verification of Security Protocols for Smart Meters in Smart Grid Networks

Mustafa Saed
Electrical and Computer Engineering
University of Detroit Mercy
Detroit, USA
email: saedma@udmercy.edu

Nizar Al Holou and Kevin Daimi
College of Engineering and Science
University of Detroit Mercy
Detroit, USA
email: {alholoun, daimikj}@udmercy.edu

***Abstract*— The smart grids' heavy reliance on cyber resources introduces frequent security concerns. The extensive attack surface revealed by the Advanced Metering Infrastructure (AMI) along with the distribution of sensitive data including; customer information, billing, and control information will provide attackers with a major economic reason to attempt the attacks. To ensure the security protocols between the various parties in smart grid are secure, automated security verification tools are sought. This paper presents one method of security verification for communications protocols between smart meters, a central gateway, and supervisory nodes using the CryptoVerif tool. There are two types of networks supported by these protocols, namely direct and indirect communication between smart meters and the central gateway. Each of these protocols has three sub-protocols: Enrollment and activation, Smart meter to central gateway security process, and key update and exchange process. The analysis of these protocols proceeds in two phases. In the first phase, the protocols were manually analyzed for security flaws, inconsistencies, and incorrect usage of cryptographic primitives. During the second phase, the protocols were analyzed using CryptoVerif, an automated formal methods-based analysis tool. Several efficiency improvements are presented as an outcome of these analyses.**

Keywords*— *Security protocol; smart meters; smart grid; formal verification; CryptoVerif.

I. INTRODUCTION

Today's smart grid networks, especially the smart meters, have more stringent security requirements compared to the traditional power grid of last decade [1]. This is primarily due to a smart grid offering a myriad amount of connections and thus is susceptible to security incidents. On top of that, new protocols are frequently introduced to take full advantage of available features and resources, such as appliances' operation schedule, incident reporting, and utilizing shared/public charging stations. Saed et al. [2] presented security protocols for smart meters within the smart grid. Before deploying any security protocol, it is prudent to do a thorough security analysis to understand the protocols' strengths and weaknesses.

The proposed protocols were designed to provide secure communications among three entities: user nodes, a central gateway, and supervisory nodes. A user node (smart meter), denoted by U, is an end-entity, typically a smart meter unit that wants to connect to a collector (central gateway). A user node may be directly connected to either a central gateway or another user node. Multiple user nodes are denoted by Ui, where i = 1 to n. The Central Gateway (collector), G, acts as a connection medium between a user node and a supervisory node (Substation). When there are multiple central gateways involved in a protocol, G1, G2… Gn are used. Finally, a supervisory node (substation), S, plays the role of a Certificate Authority (CA) in a Public Key Infrastructure. Note that the symbols and notations used in the above-mentioned protocols have been changed to serve the requirements of the verification tools.

The purpose of those communications is for the node (smart meter) to provide authenticated information, such as temperature readings and electricity consumption, to the gateways. The gateways would then bill the node based on the information received. To facilitate secure and authenticated communication between a node and a central gateway, the server (Substation) acts as a Certificate Authority that provides certificates to nodes and gateways. These protocols are expected to run over the DNPSec [3]-[4], a security framework for Distributed Network Protocol Version 3 (DNP3). DNP3 is an open and optimized protocol developed for the Supervisory Control and Data Acquisition (SCADA) Systems supporting the utility industries. Overall analysis of security protocols starts with manual analysis (phase1) followed by formal verification (phase2). In the first phase, thorough manual analysis is performed on all the protocols, with special attention given to renaming and arranging the symbols and notations used in the three protocols as these notations may not suitable for the verification algorithm. In the second phase, protocols are analyzed using a formal method-based approach. Formal methods-based verification consists of a process for generating a set of reductions that connects the protocol to some known hard problems. It is usually carried out manually, which may require a lot of creativity and effort. This could have a direct impact on the cryptographic design of the protocol. However, modern protocols tend to be more and more complicated and it is possible that a manual analysis may not cover all the aspects. Additionally, human errors may surface during the generation of long sequences of proofs. To this end, researchers sometimes resort to automated verification tools.

An excellent verification tool is CryptoVerif [5]. It is an automated tool to verify the secrecy and authenticity of a cryptographic protocol. It provides a generic mechanism for specifying the security assumptions on cryptographic primitives.

CryptoVerif has been used to analyze a number of important schemes and protocols in the field. Bhargavan et al. [6] performed a formal verification using CryptoVerif and established the correctness of the authenticated key exchange protocol within the Transport Layer Security (TLS) protocol. The authors also provided an automated proof that concluded the security of the handshake protocol within TLS is as hard as the underlying Diffie-Hellman assumption over certain groups. In the field of network security, TLS is perhaps the most important and widely used protocol for secure communications over the Internet [7]. Numerous analyses have been performed using this scheme over the years, in instances where manual analysis is performed to show the correctness of the scheme [8]-[10].

Blanchet and Pointcheval [11] analyzed the Full Domain Hash (FDH) scheme where CryptoVerif was used to generate automated security proof via sequence games. Their analysis showed that the FDH was scheme remains secure so long as the RSA scheme and the hash function are secure. FDH formalized by Bellare and Rogaway in [12] using the RSA encryption scheme [13].

Prior to introducing CryptoVerif, Blanchet [14], introduced another tool, ProVerif. This tool is enhanced and improved to get the CryptoVerif tool.

This paper adopts the CryptoVerif tool to analyze the three protocols presented in [2]. It first presents formalized descriptions of all the protocols performed. This is done by addressing some of the shortcomings identified during manual analysis, such as the translation of the notations used in the three protocols language in accordance to the language understood by cryptoVerif tool. The manual analysis also identified the potential for improvements to the protocols in terms of network throughput and latency. Next, the protocols were translated into the language (codes) of CryptoVerif for formal verification. These codes were then executed using the tool, generating a sequence of proofs that confirm the security of the protocol.

The reminder of the paper is organized as follows: Section II provides all the necessary notations and primitives. Overview of formal verification and CryptoVerif tool are presented in Section III. Section IV presents all the three protocols [2] used in CryptoVerif language. In Sections V and VI, source code for CryptoVerif and the outputs of running those codes in CryptoVerif are discussed. Section VII introduces the proposed changes. Finally, Section VIII concludes the paper with suggestions for future work.

II. NOTATIONS AND PRIMITIVES

This section will briefly introduce the notations and primitives needed for implementation of CryptoVerif. In Tables I and II depict a list of symbols and notations used.

A. *Notation:*

Protocols are described using the dot (.) notation. For example, the encryption key *ek* of user U is denoted by *U.ek*. For two bit strings x, y, x||y denotes their concatenation, and x XOR y denotes their bitwise exclusive-OR. Finally, for any two entities, such as *U* and *S*, and a message m, $U \rightarrow S : m$ denotes that *U* sends *m* to *S*.

B. *Cryptographic Primitives:*

Three cryptographic primitives: Hash Function, Public Key Encryption, and Digital Signature are described in this section. They will be used in the protocols described in Section III. Detailed definition of the notations used can be found in [14]-[15].

1) Hash Function: A cryptographic hash function family is a set of hash functions HFs, such that each HF ϵ H is a mapping from {0, 1} m to {0, 1} n, where m, n ϵ N, and m > n. The security of a cryptographic hash function is defined in terms of an adversary's ability to invert the function (one-way property), and find collisions in the functions (collision-resistance property). Detailed security definitions are omitted from this paper. There are many forms of cryptographic hash functions. The National Institute of Standards and Technology (NIST) has recommendations for some of them [16]-[17].

2) Public Key Encryption: A public key encryption, E, involves Key Generation Algorithm (EKG), Encryption Algorithm (ENC), and Decryption Algorithm (DEC), with the associated security parameter 1λ and message space M, consists of the following three probabilistic polynomial-time primitives:

a) Key Generation: $(ek, dk) \leftarrow EKG(1^{\lambda})$ Here, the input is the security parameter 1^{λ}, and the output is pair of an encryption key ek and a decryption key dk.

TABLE I. GENERAL NOTATIONS AND SYMBOLS

Symbol	**Meaning**
SCADA	Supervisory Control and Data Acquisition
DNP3	Distributed Network Protocol Version 3
U_i	User Node (Smart meter) #i, i=1, 2, …n
G	Central Gateway (Collector)
S	Supervisory Node (Substation)
Aux	Auxiliary time: Period of validity (T_1 & T_2)
CPU	Center Processing Unit
NIST	National Institute of Standard and Technology
XOR	Exclusive OR
LIST	List of all {U.ID, U.AID} Pairs
ε	Probability
f(ε)	Function of Probability

TABLE II. PROTOCOLS NOTATIONS & SYMBOLS

Symbol	Meaning
1^{λ}	Security Parameter
c	Ciphertext
U_i.ID	User Node Identity
ek	Public Key/Encryption Key
sk	Private Key or Signing Key
U_i.AID	User Node anonymous identity
$U_{i \cdot pk}$	User Node public key
$U_{i \cdot sk}$	User Node private key
G_i.ID	Central Gateway Identity
G_i.AID	Central Gateway anonymous identity
$G_{i \cdot pk}$	Central Gateway public key
$G_{i \cdot sk}$	Central Gateway private key
σ	Signature
s	Digital Signature Scheme
vk	Verification Key
$D_{SS.sk}$	Signing Key of s
$D_{SS.vk}$	Verification Key of s
$G_i.D_{SS.sk}$	Central Gatway signing key
$G_i.D_{SS.VK}$	Central Gateway verification key
U_i.r	User Node #i 's Reading, i=1, 2, …n
U_i.t	User Node #i 's Processor temperature, i=1, 2,
Hash(U_i.r)	Hash value for User Node #i 's reading, i=1, 2,
U_i.M	U_i.M = U_i.r XOR U_i.t
$S_{\cdot pk}$	Supervisory public key
$S_{\cdot sk}$	Supervisory private key
A-ID_i, A-ID_g , A-	Anonymous ID for Node, Central Gateway &
G_{cert}, $U_{i\text{–}cert}$, CR_i	Certificate of Central Gateway and User Node i
||	Concatenation
E	Public Key Encryption
→	Send to
←	Result
DEC	Decryption Algorithm of E
dk	Decryption Key of E
EKG	Key Generation Algorithm of E
ENC	Encryption Algorithm of E
H	Cryptographic Hash Function Family
HF	Hash Function
M	Message Space
m	Message
SIG	Signature Algorithm of s
SKG	Key Generation Algorithm of s
VER	Verification Algorithm of s

*b) **Encryption:*** c ← ENC(ek,m) For this notation, the input is an encryption key ek ∈ EKG(1^{λ}) and a message m ∈ M, and the output is a ciphertext c.

*c) **Decryption**:* m ← DEC(dk, c) Here, the input is a decryption key dk and a ciphertext c, where (ek, dk) ∈ EKG (1^{λ}), m ∈ M, and c ∈ ENC(ek, m).

These need a correctness criterion, which is stated as follows: for every (ek, dk) ∈ EKG(1^{λ}), m ∈ M, and c ∈ ENC(ek, m), the probability that DEC(dk, c) $\neq$ m is negligible.

The security of the encryption scheme is defined in terms of an adversary's ability to learn partial information about the message underlying a ciphertext and is based on attack models, such as Chosen-Plaintext Attack and Chosen-Ciphertext Attack [18]. There are many forms of public key encryption algorithm. NIST has recommendations for factoring-based algorithms [19] and discrete logarithm-based algorithms [20].

3) Digital Signature Scheme: A digital signature scheme, S = (SKG, SIG, VER), with the associated security parameter 1^{λ} and message space M, consists of the following three probabilistic polynomial time algorithms:

a) Key Generation: (sk, vk) SKG(1^{λ}) For this notation, the input is security parameter 1^{λ} and the output is a pair of a signing key sk and a verification key vk.

b) Signature: σ ← SIG(sk,m) For this notation, the input is a signing key sk ∈ SKG(1^{λ}) and a message m ∈ M, and the output is a signature σ

c) Verification: 1/0 ← VER(vk, m, σ) For this notation, the input is a verification key vk, a message m, and a signature σ, where (sk, vk) ∈ SKG(1^{λ}), m ∈ M, and σ ∈ SIG(sk,m). The output is 1 for valid signature, and 0 for invalid signature. These need a correctness criterion, which is stated as follows: for every (sk, vk) ϵ SKG(1^{λ}), m ϵ M, and σ ϵ SIG(sk,m), the probability that VER(vk, m, σ) $\neq$ 1 is negligible.

The security of signature schemes is defined in terms of an adversary's ability to forge a signature and is based on attack models, such as Chosen-Message Attack. There are many forms of digital signature scheme. NIST has recommendations for some of them [21].

III. OVERVIEW OF CRYPTOVERIF

This section will present provable security and formal verification required for using the CryptoVerif tool. Also, this section will explain briefly the CryptoVerif code.

A. Provable security and Formal verification

When analyzing the security of cryptographic protocols, a notion of “provable security” is raised, where adversaries are modeled as probabilistic polynomial time Turing machines with capabilities and limitations of the verification tool. Through such modeling, one can prove that the cryptographic protocol is secure by showing that if there exists an adversary under the defined model that breaks the protocol, then the certain hard mathematical problem becomes easy to solve. A proof is then obtained via contradiction: since this problem is hard to solve, then it must be that the protocol is also hard to break. A security proof for a given protocol is a reduction from the protocol to a problem assumed to be hard. This reduction can be obtained via a sequence of games, where each game is a slight modification of the previous one.

Formal verifications, or formal methods, as defined in the Dolev-Yao et al [22] framework, is a method to obtain a proof. As shown in [23], the reductions of a proof are obtained via a “sequence of games” (also known as the game hopping technique), where each game differs from previous one by being slightly changed. The sequence of games is statistically or computationally indistinguishable from the point of view of an adversary. In such a proof, the initial game is the real attack game that models the adversary and the protocol. Then, two consecutive games will look either identical, or very close to each other in the view of the adversary.

1) Rename some variables. In this case the two games are perfectly identical.

2) Replace one variable x with another one y where the distributions are “statistically” indistinguishable. Here, the two Games are statistically identical except for negligible probabilities.

3) Replace one variable x with another one y where the distributions are “computational” indistinguishable - distinguishing two Games implies the ability of solving a certain computational hard problem.

The proof is obtained via a chain of reductions such that if an adversary is capable of breaking the initial game (given protocol) with certain probability ε, then he/she is also able to break the final game with probability function of f(ε). Since the final game is assumed hard - in other words f(ε) is negligible- it is implied that ε itself is also small.

In the final game, a certain known hard problem will be arrived at in which the adversary is believed to be incapable of solving.

B. CryptoVerif

CryptoVerif is an automatic tool to generate a sequence of games. CryptoVerif can also evaluate the probability of success of an attack against the protocol as a function of the probability of breaking each cryptographic primitive and of the number of sessions (exact security). A list of reserved words of the CryptoVerif tool syntax is listed in Table III:

1) Reading the output and games

As mentioned earlier, CryptoVerif presents the results in terms of a sequence of games. As one shall be seen in Section V, the sequence is presented as follows. Using the source code, CryptoVerif performs a compilation optimization to produce the initial game, game 0. This process does not change the content of the source code. In other words, the initial game is equivalent to the source code written in CryptoVerif syntax, as it performs a sequence of reductions through the chain of the games. Only slight difference is observed between each two consecutive games. Such differences may be due to one of the following reasons:

a) Removing an unused variable. This is done with simplification pass, see VI-C for instance.

b) Removing variables that do not have an impact on the game. This is done with findcond, see VI-B for instance.

c) Applying equivalence between certain cryptographic primitive and pre-defined probability; see VI-E for instance.

Through this sequence of the reductions, the final game, which is secured by assumption, is arrived at. This completes the overall proof. If an attacker is able to break the protocol which is equivalent to the initial Game, then the attacker will also be able to break the underlying encryption algorithm with probability of at least Penccoll, or break Game 7. These are illustrated in Section VI-G.

TABLE III. CRYPTOVERIF INSTRUCTIONS

Keyword	**Meaning**
const	declare a constant
expand	expand a cryptographic primitive
fun	abstract functions
if . . . then . . .	conditional flow control
in	input to a channel
out	output to a channel
let	assign value to a variable
new	declare a variable for a given type
proba	declare a probabilistic variable
process	main function
type	define a new type of variable

2) Limitations of CryptoVerif:

CryptoVerif is a tool to generate a chain of proofs in the form of a sequence of games, to show a reduction from a given protocol into a certain hard mathematical problem or assumed-hard problem. Such a proof does not guarantee the security of the final game. Thus, if an insecure hash function, for example SHA-1 [24] is used, CryptoVerif will still produce a valid reduction to SHA-1. However, the scheme will no longer be secure due to the insecurity of SHA-1. Another limitation of CryptoVerif is that it does not find redundancy in the protocols. The redundancy occurs when extra and maybe unnecessary cryptographic operations are performed. Such operations will not reduce the overall security of the system. Hence, from CryptoVerify point of view, those operations are not flagged as they are still technically correct.

IV. SMART METER SECURITY PROTOCOL

In this section, the three protocols [2] are presented. The protocols described in sections A and B are for the direct communication setting as shown in Figure 1, and the one described in section C is for the indirect communication setting as depicted in Figure 2. The direct and indirect communications are between the user node and the central gateway.

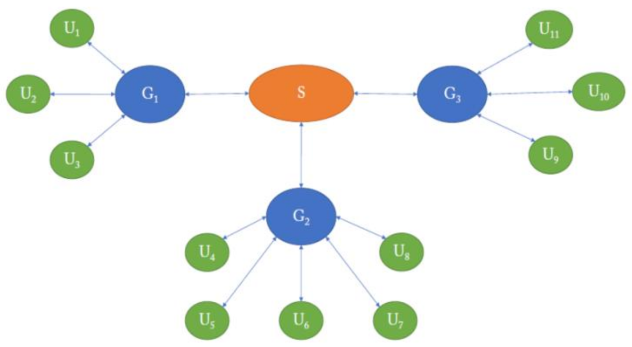

Figure 1. Direct communication.

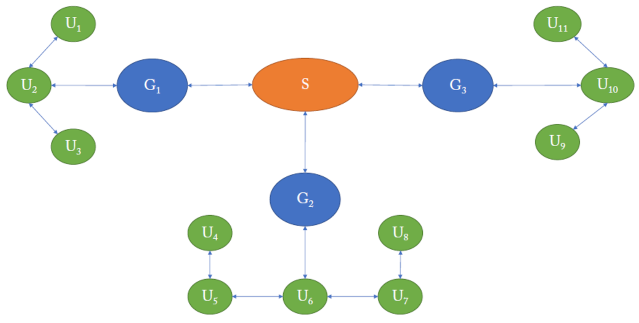

Figure 2. Indirect communication.

A. Direct communication

The first protocol consists of three processes: Enrollment and activation process, Security process, and Key update and exchange process.

1) Enrollment and Activation Process

In this process, the user nodes (smart meters) and the central gateways will be enrolled and activated so that they will be authenticated during the rest of the processes in sections IV-A-2 and IV-A-3. This process is as follows:

Step 1: The supervisory node S obtains the identities of all user nodes $\{U_i, ID\}_{i=1,2\ldots}$ and identities and encryption keys of all central gateways $\{G_i.ID, G_i.ek\}_{i=1,2\ldots}$

Step 2: $S \rightarrow \{U_i\}_{i=1,2\ldots} : \{G_i.ID, G_i.ek\}_{i=1,2\ldots}$

S simply forwards to all user nodes the identities and encryption keys of all central gateways that it obtained in step 1.

Step 3: $S \rightarrow \{G_i\}_{i=1,2\ldots} : \{U_i.ID\}_{i=12\ldots}$

S simply forwards the identities of all user nodes that were obtained in step 1 to all central gateways.

Step 4: $G \rightarrow U : (m_3\sigma_3)$, where

$m3 \leftarrow (U.ek \| G.ID)$, and

$\sigma_3 \leftarrow SIG(G.sk, m_3)$.

Here, G sends a request message and signature to U.

Step 5: U extracts G.ID from m_3 and validates it against the list obtained in step 1.

Step 6: If step 4 is successful, $U \rightarrow G : c_5 \leftarrow ENC(G.ek, m_5)$, where

$m_5 \leftarrow (U.ID \| U.ek)$.

Step 7: G decrypts c_5: $m_5 \leftarrow DEC(G.dk, c5)$, extracts U.ID from m_5 and validates it against the list obtained in step 2. If validation is successful, G extracts U.ek from m5 and stores it.

2) Security Process

Once the nodes are activated, they will be able to send relevant data to the gateways during this process.

Step 1: $U \rightarrow G : c_1$, where $c_1 \leftarrow ENC(G.ek, m_1)$. Here, $m_1 \leftarrow (U.r \| U.t \| U.m \| h)$, U.r is U's reading, U.t is U's CPU temperature, and $U.m \leftarrow (U.r \text{ XOR } U.t)$. Finally, $h \leftarrow HF(U.r)$.

Step 2: G decrypts c_1 : $m_1 \leftarrow DEC(G.dk, c_1)$, extracts (U.ID, U.t, U.m, h) from m_1, computes $U.r \leftarrow (U.t \text{ XOR } U.m)$, and checks if $h = HF(U.r)$. If the check is successful, G stores U.r.

3) Key update and exchange process

The last process of protocol A is a process for both nodes and gateways to update their keys. The process of the gateways will be shown as follows in the steps below. The process to update keys for U is similar, and therefore omitted for simplicity.

Step 1: $G \rightarrow U : (c_1 \| G.ID)$, where

$c_1 \leftarrow ENC(U.ek, G.ek \| G.vk \| \sigma)$, and

$\sigma \leftarrow SIG(G.sk, G.ek \| G.vk)$.

Step 2: U decrypts c_1: $m1 \leftarrow DEC(U.dk, c_1)$, extracts (G.ek, G.vk, σ) from m_1, and checks if $VER(G.vk, G.ek \| G.vk, \sigma) = 1$. If the check is successful, U stores G.ek and G.vk.

B. Direct communication- Certificates

The second protocol consists of three processes: Enrollment and activation process, Security process, and

Key update and exchange process. Compared to protocol A, here the major difference will be the presence of certificates.

1) Enrollment and activation process

The purpose of this process is to establish enrollments for nodes and gateways with the help of certificates. Some of the variables used below will be used in other sections preserving the same meaning.

Step1: G $\rightarrow$ S : c1 $\leftarrow$ ENC(S.ek, m1), where

$m_1 \leftarrow$ (G.AID || G.ID || G.ek)

Step2: S $\rightarrow$ G : $c_2 \leftarrow$ ENC(S.ek, $cert_2$), where

$cert_2 \leftarrow (m_2 \,||\, \sigma_2)$

$\sigma_2 \leftarrow$ SIG(S.sk, m_2)

$m_2 \leftarrow (m_1$ || AUX).

$m_1 \leftarrow$ DEC(S.dk, c_1).

Step3: U $\rightarrow$ S : $c_3 \leftarrow$ ENC(S.ek, m_3), where

$m_3 \leftarrow$ (U.AID || U.ID || U.ek).

Step4: S $\rightarrow$ U : $c_4 \leftarrow$ ENC(S.ek, $cert_4$), where

$cert_4 \leftarrow (m_4 \,||\, \sigma_4)$

$\sigma_4 \leftarrow$ SIG(S.sk, m_4).

$m_4 \leftarrow (m_3$ || AUX).

$m_3 \leftarrow$ DEC(S.dk, c_3).

Step5: S $\rightarrow$ G : LIST, where

LIST is a list of all {U.ID, U.AID} pairs.

2) Security Process

Once connections are established, this process, is adopted to exchange data between user nodes and central gateways.

Step1: U and G exchange their certificates.

Step2: Each party verifies the certificate to obtain the other party's ek and ID.

Step3: sk and ID are accepted if AUX information is also verified.

Step4: Apply the steps of A-2.

3) Certificate Update Process

This last process of protocol B allows the users and gateways to update their certificates.

Step1: S informs U and G to create new keys

Step2: Both parties will follow same steps as B-1 to get new certificates.

C. Indirect Communication

The last protocol consists of three processes: Enrollment, Activation and Certificate Exchange, Secure Reading Collection Process, and Key Update and Certificate Exchange Process.

1) Enrollment, Activation, and Certificate Exchange

This process allows user nodes and central gateways to exchange their certifications.

Step1: Ui $\rightarrow$ G : $c_1 \leftarrow$ ENC(G.ek, m_1), where

$m_1 \leftarrow$ (U_i.AID || U_i.ID)

Step2: G $\rightarrow$ U_i : $e_2 \leftarrow$ ENC(U_i.ek, $m_2 \,||\, \sigma_2$), where

$\sigma_2 \leftarrow$ SIG(G.sk, m_2)

$m_2 \leftarrow$ U_i.E.pk || U_i.AID || U_i.AUX

Step3: On node U_j for $i \neq j$: forward the message.

Step4: On node U_i:

$m_2 \,||\, \sigma_2 \leftarrow$ DEC(U_i.sk, e_2);

U_i Store m_2 and σ_2 as the message and certificate pair.

2) Secure Reading Collection Process

This process allows user nodes to send their readings to a central gateway, via either a direct connection or an indirect connection. The data flow is omitted due to the involvement of multiple entities.

Step1: Processing at node U_i:

U_i.T $\leftarrow$ TRNG;

U_i.R $\leftarrow$ reading of the data;

U_i.M $\leftarrow$ U_i.T XOR U_i.R;

m1 $\leftarrow$ U_i.M || Hash(U_i.R) || U_i.T

$\sigma_1 \leftarrow$ SIG(U_i.sk, m_1)

U_i.Y $\leftarrow$ ENC(G.ek, $m_1 \,||\, \sigma_1$)

Step2: $U_i \rightarrow U_{i-1}$: c_2, where

$c_2 \leftarrow$ ENC(U_{i-1}.ek, U_i.Y)

Step3: On node U_{i-1}:

Compute U_{i-1}.Y as in step 1;

U_i.Y $\leftarrow$ DEC(U_{i-1}.dk, c_2)

Step4: $U_{i-1} \rightarrow U_{i-2}$: c_4, where

Using a pseudonym number generation to randomize the output (c_4)

b $\leftarrow$ a random bit;

if b = 1, $c_4 \leftarrow$ ENC(U_{i-2}.ek, U_i.Y || U_{i-1}.Y)

else $c_4 \leftarrow$ ENC(U_{i-2}.ek, U_{i-1}.Y || U_i.Y)

Step5: Repeat steps 1-4 for all other user nodes until G is reached.

Step6: Processing at G:

$m_j \leftarrow$ DEC(G.dk, c_j); where j = 1 to N

Extract U_i.Y from m_j for each user;

$m_1 \,||\, \sigma_1 \leftarrow$ DEC(G.dk, U_i.Y);

Step7: Check the correctness of σ_1

Extract U_i.T , U_i.M and recover U_i.R

G Store the reading U_i.R if σ_1 is verified

3) Key Update and Certificate Exchange Process

This last process allows the entities to update their keys and the certificates associated with them.

Step1: G $\rightarrow$ U : e_1|| G.ID, where

(G.ek`, G.dk`) $\leftarrow$ KG(1^{λ})

(G.sk`, G.vk`) $\leftarrow$ KG(1^{λ})

$e_1 \leftarrow$ ENC(U.ek, $m_1 \,||\, \sigma_1$)

$m_1 \leftarrow$ G.ek` || G.sk`

$\sigma_1 \leftarrow$ SIG(G.sk, m_1)

Step2: On node U

$m_1 \,||\, \sigma_1 \leftarrow$ DEC(U.dk, e_1)

Extract G.ek` and G.vk` from m_1

If VER(G.vk, m_1, σ_1) = 1, accept G.ek` and G.vk`

Step3: U $\rightarrow$ G : e_3 || U.AID, where

(U.ek`, U.dk`) $\leftarrow$ KG(1^{λ})

(U.sk`, U.vk`) $\leftarrow$ KG(1^{λ})

$m_3 \leftarrow$ U.ek` || U.sk`

$\sigma_3 \leftarrow$ SIG(U.sk, m_3)

$e_3 \leftarrow$ ENC(G.ek, $m_3 \,||\, \sigma_3$)

Step4: On node G

$m_3 \parallel \sigma_3 \leftarrow$ DEC(G.dk, e_3)
Extract U.ek` and U.vk` from m_3
If VER(U.vk, m_3, σ_3) = 1, accept U.ek` and U.vk`

V. CRPTOVERIF CODE

The CryptoVerif code for process A-1 of protocol A (section IV-A-1) will be provided in this section. All the code sections share the same set of pre-defined parameters and macros. The sections display all the related symbols, notations, functions, and algorithms for all the related entities, namely Nodes, Gateways, and Servers, are defined using CryptoVerif language as follows:

A. *Defining hosts, nodes, gateways and server*

```
t y p e  host [ bounded ] .
c o n s t  Node : host . (* node s)
c o n s t  Gateway : host . (* gateway)
c o n s t  Server : host  . (* server)
```

B. *Defining Parameters*

The relevant macros are given as below. The spaces are the locations where the variables, such as the reading R or the time T, are drawn from.

```
t y p e Rspace [ bounded ] .
t y p e AIDspace [ bounded ] .
t y p e AUXspace [ bounded ] .
```

C. *hash function*

The hash functions, encryption functions, and decryption functions are defined. A hash function has two inputs, an input channel hc1 and an output channel hc2. The exact hash function to be used (such as SHA-2) is not defined here since only abstract functionality is required here.

```
param qH [ noninteractive ] .
channel hc1 , hc2 .
let hash oracle = ! qH i n ( hc1 , x : bit string ) ;
out ( hc2 , hash ( hk , x ) ) .
```

D. *Public key encryption parameters*

Public key, secret key (private key), seed, block size, and key generation parameters are defined below.

```
t y p e pkey [ bounded ] .
t y p e blocksize [ fixed ] .
```

```
proba Penc .
proba Penccoll .
expand IND CCA2 public key enc ( keyseed, pkey, skey, blocksize, bit string, seed, skgen, pkgen, enc, dec, injbot, Z, Penc, Penccoll).
```

E. *Digital signature algorithms*

The key locations, ciphertext spaces, and the seeds for both key generations and encryptions are defined. Then, the probability of an adversary breaking the scheme is also defined in order to achieve a meaningful reduction at the end. The encryption algorithm will be equivalent to an expansion over the above variables. The codes for signature function are as follows:

```
t y p e sskey [ bounded ] .
proba Psign .
proba Psigncoll .
expand UF CMA signature (keyseed, spkey, sskey, sblocksize, signature, sseed, sskgen, spkgen, sign, check, Psign, Psigncoll).
```

F. *Defining I/O channels*

The key spaces and ciphertext spaces are defined, as well as the seeds for both key generations and signatures. Also, as in encryption scheme, the probability of an adversary breaking the scheme is also defined, in order to achieve a meaningful reduction in the end. The signature algorithm is then an expansion over the above variables. In CryptoVerif, the variables of different types are not compatible. To solve this issue, conversion functions to resolve the incompatibility between different types are defined. To save space the conversion functions are omitted. The last step before protocol simulation is to define the input and output channels. Depending on the actual protocol, the number of channels may vary as defined below:

```
c h a n n e l c0 , c1 , c2 , c3 , c4 , c5 , start , finish .
```

G. *Defining Gateway parameters*

```
new gid : IDspace ;
new gateway seed : keyseed ;
l e t gspk = spkgen ( gateway signseed ) i n
l e t gssk = sskgen ( gateway signseed ) i n
```

H. *Defining server (substation) parameters*

```
new sid : IDspace ;
new serverseed : keyseed ;
l e t sspk = spkgen ( server signseed ) i n
l e t sssk = sskgen ( serversignseed ) i n
```

I. *Defining user node (smart meter) parameters*

```
new nid : IDspace ;
new nodeseed : keyseed ;
l e t nepk = pkgen ( nodeseed ) i n
l e t nesk = skgen ( nodeseed ) i n
```

J. *Message generation and signature*

The gateway generates a message which is the concatenation of Node's encryption key and Gateway's ID. The Gateway also signs this message.

```
l e t process step 4 =
i n ( c0 , (= nepk ,= gid ,= gssk , hostX : host ) ) ;
i f ( hos tX = Gateway ) t h e n
o u t ( c1 , (m, sigma ) ) .
```

K. *Extracting the gateway's ID*

The node extracts the Gateway's ID from the message and validates it against the list obtained in G above. It then forms another message by concatenating the ID with the Node's public key. If the code in J above is successful, the message will be encrypted with the public key of the central gateway.

```
let process step 5 6 =
in ( c1 , (msg : sblocksiz e , gidrec : IDspace , =
nid , =nepk , =gepk , hostX : host ) ) ;
if ( hostX = Node ) t h e n
new encseed : seed ;
let cipher = enc (m2 , gepk , encseed ) in
out ( c2 , cipher ) .
```

L. *Message decryption*

The Gateway decrypts the message and extracts the Node's ID.

```
let process step 7 =
in (c3, (= gesk, cipher: bitstring,=nepk,=nid,hostX: host));
if ( nidrec = nid ) t h e n
out ( c4 , ( nid , nepk ) ) .
```

M. *Assembling all process*

Finally, a master process is called to assemble all those processes

```
p r o c e s s
i n ( start , ( ) ) ;
o u t ( finish , ( nid , nepk , gid , gs sk , ge sk ) ) ;
( process step 4 | process step 5 6 | process step 7 )
```

VI. CRYPTOVERIF RESULTS

To begin with, a block diagram of the reduction flow is presented in Figure 3. Then, the details of the proofs are illustrated for section IV-A-1 as follows:

A. *Game 1- Initial State*

```
in ( s t a r t , ( ) ) ;
new gid 218 : IDspa c e ;
new gateway seed : keyseed ;
let gepk : pkey = pkgen ( gateway seed ) in
let gesk : skey = skgen ( gateway seed ) in
new serversignseed : skeyseed ;
let sspk : spkey = spkgen ( serversignseed ) in
let sssk : sskey = sskgen ( serversignseed ) in
new nid : IDspace ;
new nodeseed : keyseed ;
if ( hostX 220 = Gateway ) t h e n
new sigseed : sseed ; 23
let m: sblocksize = concat 5 ( gid 218 , nepk 219 ) i n
let sigma : signature = sign (m, gssk , sigseed ) in
out ( c1 , (m, sigma ) )) j (
in ( c1 , (msg : sblocksize , gidrec : IDspace , =
nid , =nepk 219 , =gepk , hostX 221 : host ) );
if ( hostX 221 = Node ) t h e n
let concat 5 ( gid 223 : IDspace , nepk 222 : pkey ) = msg in
if ( gid 223 = gidrec ) t h e n
let m2 : blocksize = concat 4 ( nid , nepk 222 ) in
new encseed : seed ;
let cipher 2 2 4 : bitstring = enc (m2 , gepk ,encseed ) i n
out ( c2 , cipher 2 2 4 ) ) j (
in ( c3 , (= gesk , cipher 2 2 6 : bitstring , =
nepk 219 , =nid , hostX 225 : host ) ) ;
if ( nidrec = nid ) t h e n
out ( c4 , ( nid , nepk 219 ) )
```

B. *Difference between Game 1 and 2*

In game 2, the find condition function, findcond, is applied to remove the assignments for the following variables in the code: gspk, sepk,sesk, sspk,sssl and nesk. These are the variables that were not used in the game but necessary during the execution. For example, the public key for the central gateway gspk was not used, but since its secret key is crucial to the proof, it is still necessary to generate this key for the protocol. To sum up, this modification yields the removal of the following lines of code:

```
l e t gspk : spkey = spkgen ( gateway signseed ) in
new sid : IDspace ;
l e t sssk : sskey = sskgen ( serversignseed) in
l e t nesk : skey = skgen ( nodeseed ) in
```

C. *Difference between Game 2 and 3*

In Game 3, a simplification process is performed to remove the following lines:

```
new sid : IDspace ;
new serverseed : keyseed ;
new serversignseed : skeyseed ;
```

D. *Difference between Game 3 and 4*

In Game 4, the tool tries to remove assignments for the variable nepk. This yields:

```
- out ( f i n i s h , ( nid , nepk 241 , gid 240 , gssk ,
gesk ) ) ;
+ out ( finish , ( nid , pkgen ( nodeseed ) , gid 240 ,
gssk , gesk ) ) ;
in ( c0 , (= nepk 241 , =gid 240 , = gssk ) ) ;
+ in ( c0 , (= pkgen ( nodeseed ) , =gid 240 , = gssk ) );
l e t m: sblocksize = concat 5 ( gid 240 ,
nepk 241 ) i n
+ l e t m: sblocksize = concat 5 ( gid 240 , pkgen (
nodeseed ) ) in
in ( c3 , (= gesk , cipher 2 4 5 : bitstring , =
nepk 241 , = nid ) ) ;
+ in ( c3 , (= gesk , cipher 2 4 5 : bitstring , =pkgen
( nodeseed ) , = nid ) ) ;
out ( c4 , ( nid , nepk 241 ) )
+ out ( c4 , ( nid , pkgen ( nodeseed ) ) )
```

E. Difference between Game 4 and 5

The formal reductions are performed in Game 5, where the tool applies the following:

IND CCA2(enc) with node seed, encseed = probability Penccoll + Penc(time(context for Game 4) + time, 0). This equation indicates the security of the underlying encryption primitive, depending on the ciphertext indistinguishability against chosen ciphertext attacks (IND-CCA), which is a function of the probability of finding collisions (Penccoll) and the time of encryption (Penc). With a few substitutions for some of the variables, this yields:

```
-  new nodeseed : keyseed ;
-  out ( finish , ( nid , pkgen ( nodeseed ) , gid 240 ,
- new encseed : seed ; 11
- l e t cipher 2 4 4 : bitstring = enc (m2 , gepk ,
encseed ) in
+ l e t @8 x 285 : blocksize = m2 i n

+ l e t @8 y 284 : pkey = gepk i n

+ l e t cipher 2 4 4 : bitstring = enc (@8 x 285 ,

@8 y 284 , @8 r3 287 ) in
```

F. Difference between Game 5 and 6

During this step, the individual variables for public keys and ciphertexts are removed. Their values are saved. These variables are passed directly to the encryption function for compactness.

```
- l e t @8 x 285 : blocksize = m2 in
- l e t @8 y 284 : pkey = gepk in
- l e t cipher 2 4 4 : bitstrin g = enc (@8 x 285 ,
@8 y 284 , @8 r3 287 ) in
+ l e t cipher 2 4 4 : bitstring = enc (m2,gepk,@8 r3 287 ) in
```

G. Difference between Game 6 and 7

Since the encryption function in previous step has been made explicit without referring to

run-time variables, there is no need to use an encryption seed anymore. This is reflected in this code below:

```
new encseed 286 : seed ;
```

Once Game 7 is reached, a conclusion is generated via the following inequality:

ADV(Game1; Initial Game) $\leq$ Penccoll + Penc(time(Game 4) + time, 0) + ADV(Game7; Initial Game)   (1)

This implies that the Advantage an adversary gains to be able to break the initial Game is less than or equal the sum of three quantities; the probability to break the encryption scheme Penccoll, the time to perform various polynomial time algorithms such as key generation and encryption Penc(time(Game 4) + time, 0), and the Advantage of breaking the last Game by the adversary ADV(Game7; Initia Game). By assumption, this last quantity is bounded by:

ADV(Game7; Initial Game) $\leq$ 0   (2)

The last two quantities are both small. Hence, if the adversary is able to break the IV-A-1 process, Penccoll must be non-negligible, which implies that the adversary must also be able to break the underlying encryption algorithm.

H. Summary of Statistical Analysis

The summery of the statistical analysis for the three protocols and the number of Games that is required to obtain a proof is presented in table IV. The following equations are used in the statistical analysis

Pre-compile optimization = 1 – (No. of lines in the initial Game/No. of lines of codes)   (3)

Total run time optimization = 1 – (No. of lines in the final game/No. of lines in the initial Game)   (4)

Average run time optimization = Total run time optimization/Number of Games   (5)

Total improvement = 1 – (No. of lines in the final Game/No. of lines of codes)   (6)

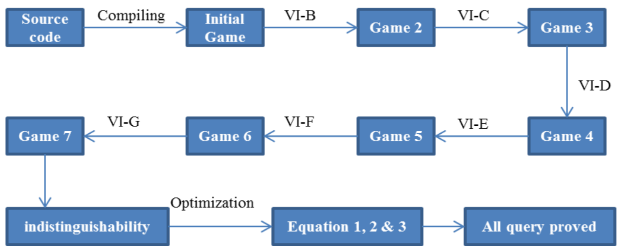

Figure 3. Reduction flow for A-1.

TABLE IV. GAMES SATATISTICS FOR THE PROTOCOLS

	Section								
Variables	A-1	A-2	A-3	B-1	B-2	B-3	C-1	C-2	C-3
Number of Games	8	3	11	7	3	7	3	3	17
Line of codes	177	243	185	186	247	196	182	258	238
Line in the Initial Game	40	49	39	50	50	50	49	60	66
Line in the Final Game	30	40	32	46	40	46	46	56	60
Pre-Compile Optimization (Eq.3) %	77.4	79.8	78.9	73.1	79.7	74.5	73.1	76.7	72.3
Run time optimization (total) (Eq. 4) %	25	18.4	17.9	8.0	20	8.0	6.1	6.7	9.1
Average run time optimization (Eq.5) %	3.13	6.13	1.63	1.14	6.67	1.14	2.04	2.23	0.53
Total improvement (Eq. 6) %	83.1	83.5	82.7	75.3	83.8	76.5	74.7	78.3	74.8

VII. PROPOSED CHANGES AND IMPROVEMENTS

In addition to the automated analysis from CryptoVerif, a preliminary manual analysis was also performed and the following findings are observed. First, protocol A does not state how authenticity was established. It was assumed that entities involved in this protocol obtain certain mutual authenticity through some channel not specified by the protocol. Inappropriate authentication methods may lead to potential attacks. Nonetheless, it is common that in practice one may rely on pre-established authenticity. It is also worth noting that in protocol B and C, certificates are used for authentication. Second, some of the operations seem to be redundant.

For example, in protocols A and B, signatures are encrypted before they are sent. Thus, this operation does not add any additional security features to the protocol. Third, in protocol C, the forward message employs a random bit to determine the sequence of concatenating the encrypted readings Y_{i-1} and Y_i. This design is adhoc and a security proof for this is not straightforward. Nonetheless, it is noted that this operation at least will not reduce the overall security.

VIII. CONCLUSION AND FUTURE WORK

In this paper, a formal analysis of three security protocols for the communication of smart meters was performed using CryptoVerif. A proof is produced through the reduction of a sequence of games. The analysis showed that the three schemes have no flaws in their security. In addition, a manual analysis of the protocol was performed and some redundancy was observed. This redundancy does not impact the overall security of the protocol, but it is highly likely that the protocol will be more efficient once the redundancy is removed.

The security evaluation performed with CryptoVerif revealed that the protocol is secure. A logical next step would be to perform simulation and benchmarking to judge the efficiency of the protocols. Once the protocols reach certain level of maturity, the next straightforward future work will be the deployment of the three protocols.

REFERENCES

[1] Draft Smart Grid Cyber Security Strategy and Requirements, NIST IR 7628, Sept, 2009

[2] M. Saed, K. Daimi, and N. Al-Holou, “Approaches for Securing Smart Meters in Smart Grid Networks,” International Journal On Advances in Systems and Measurements, pp. 265–274, 2017.

[3] M. Majdalawieh, F. Parisi-Presicce, and D. Wijesekera, DNPSec: Distributed Network Protocol Version 3 (DNP3) Security Framework. Dordrecht: Springer Netherlands, 2006, pp. 227–234. [Online]. Available: https://doi.org/10.1007/1-4020-5261-8 36, [retrieved: May, 2018].

[4] TriangleMicroWorks, “Modbus and DNP3 Communication Protocols,”2017.[Online].Available: https://scadahacker.com/library/Documents/ICSProtocols/TriangleMicroworks-Modbus-DNP3Comparison.pdf, [retrieved: May, 2018].

[5] B. Blanchet, “CryptoVerif: Cryptographic Protocol Verifier in the Computational Model,” INRIA, Tech. Rep., 2017. [Online]. Available: http://prosecco.gforge.inria.fr/personal/bblanche/cryptoverif/, [retrieved: May, 2018].

[6] K. Bhargavan, B. Blanchet, and N. Kobeissi, “Verified models and reference implementations for the TLS 1.3 standard candidate,” in 2017 IEEE Symposium on Security and Privacy, SP 2017, San Jose, CA, USA, May 22-26, 2017, 2017, pp. 483– 502. [Online]. Available: https://doi.org/10.1109/SP.2017.26, [retrieved: May, 2018].

[7] T. Dierks and E. Rescorla, The Transport Layer Security (TLS) Protocol Version 1.2. Internet Engineering Task Force, 2008. [Online]. Available: https://tools.ietf.org/html/rfc5246, [retrieved: May, 2018].

[8] B. Dowling, M. Fischlin, F. Gnther, and D. Stebila, “A cryptographic analysis of the tls 1.3 handshake protocol candidates,” Cryptology ePrint Archive, Report 2015/914, 2015, https://eprint.iacr.org/2015/914, [retrieved: May, 2018].

[9] X. Li, J. Xu, Z. Zhang, D. Feng, and H. Hu, “Multiple handshakes security of tls 1.3 candidates,” in 2016 IEEE Symposium on Security and Privacy (SP), May 2016, pp. 486–505.

[10] C. J. F. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. van der Merwe, “A comprehensive symbolic analysis of tls 1.3,” in CCS, 2017, pp. 1773-1788.

[11] B. Blanchet and D. Pointcheval, “Automated security proofs with sequences of games,” in CRYPTO’06, ser. Lecture Notes in Computer Science, C. Dwork, Ed., vol. 4117. Santa Barbara, CA: Springer, Aug. 2006, pp. 537–554.

[12] M. Bellare and P. Rogaway, “The exact security of digital signatures-how to sign with rsa and rabin,” in Advances in Cryptology — EUROCRYPT ’96, U. Maurer, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 399–416.

[13] R. L. Rivest, A. Shamir, and L. M. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” Commun. ACM, vol. 21, no. 2, pp. 120–126, 1978.

[14] B. Blanchet, “Automatic verification of security protocols in the symbolic model: the verifier ProVerif,” in Foundations of Security Analysis and Design VII, FOSAD Tutorial Lectures, ser. Lecture Notes in Computer Science, A. Aldini, J. Lopez, and F. Martinelli, Eds. Springer, 2014, vol. 8604, pp. 54–87.

[15] J. Katz and Y. Lindell, Introduction to Modern Cryptography, Second Edition, 2nd ed. Chapman & Hall/CRC, 2014.

[16] S. Goldwasser and M. Bellare, “Lecture Notes on Cryptography,” MIT, Tech. Rep., 2008. [Online]. Available: https://cseweb.ucsd.edu/_mihir/papers/gb.pdf, [retrieved: May, 2018].

[17] Q. H. Dang, Secure Hash Standard (SHS). National Institute of Standards and Technology, 2015. [Online]. Available: https://dx.doi.org/10.6028/NIST.FIPS.180-4, [retrieved: May, 2018].

[18] NIST, SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions. National Institute of Standards and Technology, 2015. [Online]. Available: http://doi.org/10.6028/NIST.FIPS.202, [retrieved: May, 2018].

[19] M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway, “Relations among notions of security for public-key encryption schemes,” in CRYPTO, 1998, pp. 26–45.

[20] E. Barker, L. Chen, and D. Moody, Recommendation for Pair-Wise Key-Establishment Schemes Using Integer Factorization Cryptography. National Institute of Standards and Technology, 2014. [Online]. Available: http://doi.org/10.6028/NIST.SP.800-56Br1, [retrieved: May, 2018].

[21] E. Barker, L. Chen, A. Roginsky, and M. Smid, “Recommendation for Pair-Wise Key Establishment Schemes Using Discrete Logarithm Cryptography,” National Institute of Standards and Technology, 2013. [Online]. Available: http://doi.org/10.6028/NIST.SP.800-56Ar2, [retrieved: May, 2018].

[22] E. Barker, “Digital Signature Standard (DSS),” National Institute of Standards and Technology, 2013. [Online]. Available: http://doi.org/10.6028/NIST.FIPS.186-4, [retrieved: May, 2018].

[23] D. Dolev and A. C. Yao, “On the security of public key protocols,” IEEE Trans. Information Theory, vol. 29, no. 2, pp. 198–207, 1983. [Online]. Available: https://doi.org/10.1109/TIT.1983.1056650, [retrieved: May, 2018].

[24] M. Bellare and P. Rogaway, “The security of triple encryption and a framework for code-based game-playing proofs,” in Advances in Cryptology - EUROCRYPT 2006, S. Vaudenay, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 409–426.

[25] X. Wang, Y. L. Yin, and H. Yu, “Finding collisions in the full SHA-1,” in Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings, 2005, pp. 17–36. [Online]. Available: https://doi.org/10.1007/11535218 2, [retrieved: May, 2018].