electronics

Article

Securing Publisher–Subscriber Smart Grid Infrastructure

Fraser Orr [1], Muhammad Nouman Nafees [1], Neetesh Saxena [1,*] and Bong Jun Choi [2]

[1] School of Computer Science & Informatics, Cardiff University, Cardiff CF10 3AT, UK; orrfm@cardiff.ac.uk (F.O.); nafeesm@cardiff.ac.uk (M.N.N.)
[2] School of Computer Science and Engineering, Soongsil University, Seoul 06978, Korea; davidchoi@soongsil.ac.kr
* Correspondence: nsaxena@ieee.org

Abstract: The security of communication protocols in the smart grid system is a crucial concern. An adversary can exploit the lack of confidentiality and authentication mechanism to cause damaging consequences. In the substation automation systems that rely on multicast communication between various intelligent electronic devices, the lack of security features in the standard IEC61850 and IEC62351 can invite attackers to manipulate the integrity of the employed publisher–subscriber communication paradigm to their advantage. Consequently, many researchers have introduced various approaches offering authenticity and confidentiality. However, such schemes and methods for the aforesaid standards have computational limitations in compliance with the stringent timing requirements of specific applications in the smart grid. In this paper, we propose an approach that can fully secure the publisher–subscriber communication against confidentiality attacks. In this direction, we develop a demo tool to validate the performance of our proposed security approach for potential factors such as timing requirements and the size of the messages. Finally, we evaluate our scheme considering the requirements of the GOOSE, SMV, and MMS protocols in the substation automation systems.

Keywords: smart grid; security; publisher–subscriber model

Citation: Orr, F.; Nafees, M.N.; Saxena, N.; Choi, B.J. Securing Publisher–Subscriber Smart Grid Infrastructure. *Electronics* **2021**, *10*, 2355. https://doi.org/10.3390/electronics10192355

Academic Editor: Jahangir Hossain

Received: 27 July 2021
Accepted: 22 September 2021
Published: 27 September 2021

1. Introduction

Power grid networks are becoming an evolutionary step of providing electricity in long distances; the now digitized paradigm has improved its control and performance capabilities. To realize digitized information, substation automation of the grid uses data from sensors and intelligent electronic devices (IEDs) to control remote power system processes [1]. According to a report published in [2], the global substation automation market size is estimated to be USD 39.9 billion in 2021. It is projected to rise to USD 54.2 billion by 2026 due to several prominent factors, including increasing power grid development projects. Substation communication is an integral part of the reliable operation of the power grid. The substation follows unicast, multicast, and broadcast communication messages for different purposes. For example, the publisher–subscriber paradigm is employed in multicast communication within a power substation communication for instant event notification and asynchronous parallel processing among various control components, such as intelligent electronic devices (IED) [3]. In this context, the Phasor Gateway (PG) connects to the network through status routers as a publisher, whereas data destinations are generally the power usage vendors that act as subscribers in the publisher–subscriber paradigm. This multicast communication model allows faster response and reduces the delivery latency by eliminating the need to periodically "poll" for new updates in the substation automation settings.

Trust in the underlying communication system for the power grid's control process applications is of paramount importance. However, security features of the communication protocols used in substation automation are not included in IEC61850 due to the trade-off with the following factors: (1) high-speed performance requirement; and (2) limited computational resources of the deployed IEDs. Moreover, security was not the primary concern when the IEC61850 was initially published [4]. However, trusting the communication protocol commands without any validation is not an effective trade-off in a power grid system; high capability adversaries can take advantage of intrinsic communication protocol vulnerabilities to perform stealthy attacks to impact the control processes of the smart grid.

Given the lack of security mechanisms in industrial communication protocols, it is always possible for an adversary to mount security attacks [5]. Attacks such as man-in-the-middle (MITM), malicious control command injection, replay, and information tempering can cause severe damage to the smart grid operations and services [4]. For example, if an attacker gets access to the network, he can mount information tempering attacks by targeting the publishers to impact its operations maliciously. Similarly, an attacker can compromise the monitoring port of the process bus Ethernet switch to obtain real-time sampled values (SV) messages; they can then mount a replay attack by passing on previous SV packets that contain fault current and voltages values [1]. The attacker can leverage this attack to open the circuit breakers by manipulating the SV subscribers.

To protect the communication protocols against malicious actors, its integrity, confidentiality, and authenticity must be assured. Towards this end, the smart grid network must have a secure communication mechanism to ensure that the control commands are sent and received by legitimate communication nodes in the system [6]. IEC 61850 recommends some specific security solutions for multicast messages: generic object-oriented substation events (GOOSE) and sampled measured value (SMV) should use digital signatures with Rivest–Shamir–Adleman (RSA) algorithms to verify message integrity and authenticity [7]. However, it has been observed that the generation of digital signatures with RSA algorithms entails high computational resources in few milliseconds. In addition, other solutions such as elliptic curve digital signature algorithm (ECDSA)-based signatures require long processing times, particularly when it is applied to GOOSE communication [7]. For example, IEC 61850-5 recommends the latency requirement of message delivery within four milliseconds (ms). Therefore, to accommodate such stringent latency requirements, a realistic and flexible security solution must be proposed to secure the publisher–subscriber model in the substation automation of the smart grid. Moreover, the security solution should be based on a simple algorithm with limited computational power and few modifications that support existing applications.

Contributions. In this paper, we present a secure approach scheme for multicast communication in substation automation systems. Specifically, we focus our work on addressing security issues in the publisher–subscriber communication model. The idea is to propose a solution that conforms to the stringent timing requirements for various communication protocols such as GOOSE, SMV, and Manufacturing Message Specification (MMS). In this direction, we perform various experiments by modifying the structure of messages: For example, we test the different sizes of messages and private keys. Our main contributions are as follows:

1. We develop a tool to investigate and evaluate the performance of our approach that incorporates encryption algorithms with various potential factors such as the size of the messages.
2. We implement the encryption algorithms as per IEC61850 to secure GOOSE, SMV, and MMS communication in the substation automation systems.
3. We analyze the efficacy of our proposed approach in evaluating it concerning the specific requirements of the protocols in the substation automation systems.

The rest of the paper is organized as follows. Section 2 describes the communication system in the context of a multicast communication system in the smart grid, and it also discusses an adversary model for this paper and the related works. Section 3 presents the proposed system and our approach. We explain the system design and present an implementation of a security scheme for multicast communication in the smart grid. Section 4 provides the results and evaluation of our proposed scheme along with attacks analysis. Finally, Section 5 concludes the work.

2. Communication and Adversary Models

This section introduces the system modeling in substations and presents the adversary model for this work.

2.1. Substation Automation Features

An electrical substation consists of bays includes subparts in the substation, usually with common functionalities. The substation controls critical processes such as protective relay operations through communication networks [8]. For example, the transformer and circuit breakers work cooperatively along with the functions of protective relays to transform voltages and protect transmission lines and feeders. This cooperative operation combining various functions also forms a bay named the transformer bay.

An essential function of substation includes monitoring of control processes. Several intelligent electronic devices (IEDs) are connected to power devices that usually have an inbuilt analog-to-digital converter (ADC). The IEDs deliver digitized messages about their running states to the other devices in the system. For example, a merging unit (MU) IED delivers digitized current and voltage signals to protect and control IED using SV messages.

2.2. Communication Model

As shown in Figure 1, a general communication system architecture consists of a communication protocol, where different multicast messages are transmitted among different devices. For example, protection messages are transmitted between relays through the station bus, whereas SV messages on the process bus are transmitted to several IEDs at the bay level from the merging units. Table 1 summarizes the time criticality of various multicast messages for different substation applications, including control and protection messages. An adversary can modify these messages if authentication is not provided. Towards this end, three communication protocols are mainly used for various substation applications: MMS, GOOSE, and SMV. In this direction, MMS follows the client–server communication model, whereas GOOSE and SMV follow the subscriber–publisher communication model [9].

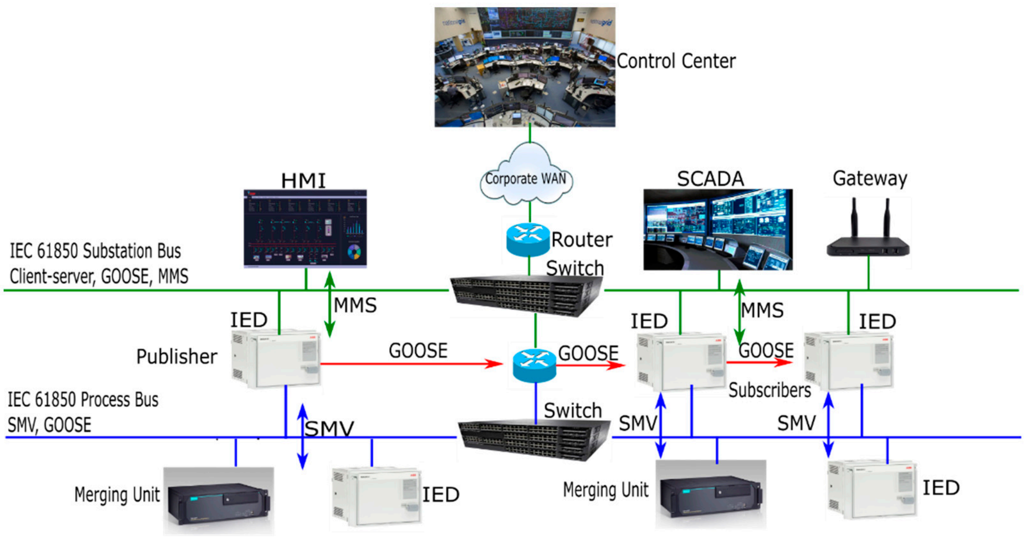

Figure 1. Smart grid system and protocols.

Table 1. Multiclass communication requirements in smart grid.

ID	Message Type	Requirement Name	Criteria	Comment
R1	GOOSE, SMV	Time Constraint	The total time from publisher encryption to the message being decrypted by the subscriber should take no longer than 4 ms	This is the defining requirement for the algorithm is the time complexity will dictate what solutions should be discussed
R2	GOOSE, SMV	Model	The system should support multicast communication for the publisher–subscriber model	This is to emulate the real-world communication the smart grids use to communicate
R3	MMS	Time Constraint	Time is less critical with total time ranging from 100–500 depending on if it's a low/medium or command message	These are not very quick time conditions so a wide range of algorithms should be applicable
R3	MMS	Model	Follows a client–server model	This is a more standard model to implement

When analyzing the message types being used, i.e., GOOSE, SMV, and MMS, it is essential to distinguish them based on their different design criteria and topologies. This means that one design approach may not cover all the criteria of all message types in all scenarios. Furthermore, it can be seen from Table 1 that the primary considerations for each message type are the time requirements and the model that the message type supports; this shows that the problem is split into the GOOSE/SMV side and the MMS side as each of have different considerations. In addition, it is also an essential requirement to consider the communication of such protocols with the devices. For example, when designing any encryption algorithm, the non-functional requirements with the IEDs should be considered. Table 2 summarizes the non-functional requirements concerning the algorithms.

Table 2. Non-functional requirements for encryption algorithm.

ID	Requirement Name	Priority	Criteria	Comment
NF1	Algorithms must be lightweight	High	To be deployed to IEDs, the algorithms must be able to run in a limited environment	This directly ties into how quickly the algorithm runs with most fast encryption methods based on lightweight methods
NF2	Time constraints	High	The specification for message types must be met	Any implementation must meet the constraints for each message type, as mentioned previously
NF3	Message Integrity	High	The message sent on any model must be able to be decoded and read by the recipient	All encryptions must be able to be decrypted by the intended recipient
NF4	Attack prevention	High	All encryption algorithms used must not be susceptible to any known attack	The implementation must be cryptographically secure from known attacks such as Man-in-the-middle or replay attacks
F1	User Comparison	High	It should be easy for the user to compare different algorithms with different message types	Using the tool, it should be easy to look at the results of

Publisher—Subscriber Model. The publisher–subscriber model is a messaging pattern where the publishers (senders) publish messages into the communication infrastructure, and the subscribers (receivers) express interest in a particular message category [10]. This is very different from the synchronous request-response model and is a much more scalable solution due to no limitations surrounding centralized data.

Within the IEC 61850 framework, GOOSE messages and messages transmitting sampled values (SV) are the main types of messages that require indirect asynchronous delivery. The publisher–subscriber model can take advantage of multicast messaging, which allows the sending of a single copy that will be replicated and passed on throughout routers and forwarded to subscribers that have previously signaled interest. The communication infrastructure is responsible for the delivery of the messages and maintains the subscription information.

Publisher–subscriber architecture is vulnerable to man-in-the-middle (MITM), replay, and impersonation attacks. The system also suffers from publisher (sender) authentication, subscriber (receivers) authorization, and data integrity issues. The working of the publisher–subscriber model and how messages are communicated are discussed in [3].

Manufacturing Message Specification (MMS). The manufacturing message specification (MMS) is an ISO 9506 standard that is used to transfer real-time process data and control information between the network devices, such as an IED and the HMI application running on a PC. It follows a more traditional client–server model for communication. MMS has the slowest time requirements [11], compared to the publisher–subscriber model with a broader range of data. Therefore, it has the most flexibility when using an encryption algorithm as it does not need to be as lightweight.

Generic Object-Oriented Substation Event (GOOSE). Various services in substation employ GOOSE in the publisher–subscriber model, such as a generic substation event, i.e., a control model defined as per IEC 61850 responsible for the fast mechanism of transferring event data across substation network. Towards this end, the publisher multicast the written values in a transmission buffer to different subscribers. The power quality monitoring devices are usually interested in the GOOSE messages published by the LEDs. To receive such messages, the devices need to subscribe to the published devices in the publisher–subscriber paradigm.

GOOSE supports exchanging a wide range of possible common data organized by a DATA-SET. GOOSE messages are used to replace the hard-wired control signal exchange between various IEDs for interlocking, protection purposes, sensitive missions, high reliability, and time criticality [12]. GOOSE messages are exchanged at the data link layer using the multicast functionality of ethernet cables. When triggered by a preconfigured event, an IED sends the GOOSE message containing values for the variables that need to be communicated, such as carry monitoring and control functions, tripping, and interlocking information. Since these are multicast messages, there is no acknowledgment mechanism. Moreover, GOOSE messages are generally used within the substation automation systems to carry the breaker trip or close commands. However, the confidentiality of the messages becomes more significant when GOOSE is used to communicate with DERs for energy management or market purposes.

Therefore, the GOOSE messages must be encrypted for confidentiality purposes. However, the algorithm to encrypt such messages must conform to the stringent GOOSE time requirements of 4 ms [13]. Towards this end, a short GOOSE message handling just one digital status information in its dataset has an approximate size of 124 bytes. The actual size depends on various configured parameters in the GOOSE control block, such as GoID, the name of the dataset, and the reference object of the GOOSE control block. The typical size of GOOSE messages is between 92 bytes to 250 bytes [14].

Sampled Measured Values (SMV). IEC 61850-9-2 defines the sampled measured values traffic, which carries voltage and digital current samples. Towards this end, the merging units (MUs) receive three-phase current signals from various transmission lines and convert them to SMV messages over an Ethernet network. SMV also follows a publisher–subscriber model utilizing multicast messaging. For example, MU IED becomes a publisher when SMV messages are transmitted between different IEDs while the control IED serves as subscribers [1]. In addition, the SMV protocol uses OSI model Layer 2 (data link) for communication identified by MAC address and the identifier in the message body [14]. Therefore, plain text communication of SMV at the data link layer reveals critical data information of control processes in the communication network. An adversary with access to the process bus can extract useful information from the semantic of SMV messages. For example, the adversary can modify the three-phase current and voltages values received by MU to activate the safeguard scheme at the control IEDs.

2.3. Adversary Model

Generally, the adversary can be anyone capable of performing security attacks over an insecure network. In addition, adversaries who know the publisher–subscriber architecture of substation automation systems can mount MITM, information tempering, and replay attacks. Furthermore, actors who perform false data injection attacks require sufficient information power system dynamics to keep the attacks stealthy. In contrast to stealthy false data injection attacks, no expertise in power systems is required for other types of attacks against multicast communication protocols in substation automation systems.

An adversary can access a process bus in a substation to monitor the communication traffic to examine protocols' semantics; however, physical access to the process bus communication network is not always necessary. Therefore, we assume that the adversary remotely exploits backdoor access to the substation local operating network on a bay level which connects various IEDs and enables protection applications. Communication over the local operating network in the substation is not encrypted for IEC 61850 traffic; attackers can mount attacks, such as replay, false data injection, and MITM attacks. An adversary can leverage this attack for reconnaissance purposes; he/she can monitor communication traffic to identify GOOSE and SV messages. For example, an adversary can precisely monitor the smpCnr field, which increments in line with the transmission of data packets. Next, an attacker can modify the voltage measurement and replay the older fault current and voltages measurements to affect the relay protection functions of the SV subscribers. The attack's impact can manipulate the functions of the IEDs due to the multiple data streams of measurements; one data stream from the MU and the other the replayed packets. Hence, the normal operations of the IED may get blocked. Consequently, a delay in fault clearance by the IEDs may occur in the event of overloaded lines or short-circuits. In the worst-case scenario, this attack can overload the transmission lines with the protective relay being blocked, potentially triggering cascading failures in power systems and blackouts.

Similarly, an adversary can inject false measurements with abnormal information. To mount this attack, the adversary can utilize the information from the reconnaissance stage to spoof GOOSE messages. In this direction, an attacker can modify the GOOSE data payload that issues trip signals and contains sequence number fields. The attack can be conducted at a higher rate to mimic relay tripping; in so doing, the attacker can cause the circuit breakers to open, which can trigger tripping of transmission lines and under-frequency load-shedding in the worst-case scenario. In addition, a malicious connection between the publisher and subscriber can be created to perform a MITM attack by an adversary. By doing so, an adversary can impersonate the publisher or the subscriber to leverage the attack to introduce critical control function faults.

It is important to note that, while the aforementioned attacks are initiated mostly as communication attacks, the ability of the adversaries to leverage the attack to influence the physical processes of the smart grid is improving. Moreover, these attacks can be mounted to cause sudden impact; high capability adversaries can accomplish a high level of success by utilizing these attacks stealthily over a more extended period as part of an advanced persistent threat campaign against the infrastructure.

While the primary motivation of our work is the growing sophistication of attacks against multicast insecure communication messages, our proposed solution assumes a data integrity attack, which can also be accomplished remotely. In this direction, we assume that the adversaries can manipulate the critical values in communication data packets, replace a legitimate command, or inject a malicious command. Apart from these capabilities, we assume that the attackers have knowledge of power and communication system topology.

Moreover, the attackers have also required resources and have partial access to the system to perform integrity attacks.

Our approach aims to prevent these attacks; each control command issued in the publisher–subscriber communication paradigm must undergo the lightweight encryption process. Moreover, any attempt by an adversary to mimic a subscriber and mount a MITM attack would trigger an attack detection alarm; each publisher would have a subscribers list, and an encryption key would be required to become a subscriber. In this direction, an attempt to communicate from outside the network automatically raises a detection alarm.

Implementation of our solution needs to be carried out as part of a limited computing resource consideration. For example, we assume that adversaries cannot perform flooding attacks, causing an increase in the computation overhead while mounting a MITM attack or any similar attacks.

2.4. Related Work

Several solutions for the security of communication protocols in the smart grid have been proposed in the literature. Furthermore, Refs. [14,15] analyzed the conformed test on IEC 61850 standard; however, security solutions for the multicast messages were not the scope of the research. Falk [16] proposed a security solution for GOOSE and SMV packets. In this direction, the authors proposed an authentication mechanism for the aforementioned protocols. However, practical details concerning latency requirements for specific applications were not discussed. To address security concerns in the publisher–subscriber model, Fateri et al. [17] presented a simulation-based traffic analysis; however, stringent timing requirements of the protocols were not discussed. Recently, the performance of security features enabled secured sampled value packets transmitted between protection and control devices was discussed in [1]. Towards this end, the authors proposed a prototype on a low-cost embedded system for MAC-enabled SV messages to fully secure the process bus communication of the substation automation systems. However, other security implementations on other protocols, such as GOOSE and MMS, were not covered.

3. Proposed Approach

In this section, we present our proposed system design and discuss our approach.

3.1. Approach Idea

To facilitate the understanding of the proposed tool's interaction with the system, our proposed approach is shown in Figure 2. Towards this end, the basic flow of the network for the system model alongside the data exchange is also shown. The sender and receiver are treated as two nodes in a network, with communication between the nodes handled by the network class to simulate communication over a network. In this direction, we consider the network to be insecure, so all communication between the nodes should be encrypted. For the encryption, the nodes will have access to the encryption methods as they should handle the entire process with no help other than the transmission of the ciphertext and keys, which the network would handle.

3.2. Proposed Scheme

We propose a new secure scheme for a power substation, based on publisher–subscriber architecture. The proposed scheme maintains confidentiality and message integrity. In the original model of the smart grid publisher–subscriber model, a publisher multicasts a data message to all its subscribers. Upon receiving the message, each subscriber can decode the message and verifies message integrity. In the traditional publisher–subscriber model, these messages are sent as plaintext and their integrity and confidentiality are not verified. Clearly, an adversary can alter the transmitted message over the insecure network. Referring to Table 1, the proposed scheme aims to offer integrity and confidentiality to transmitted messages based on their latency requirements.

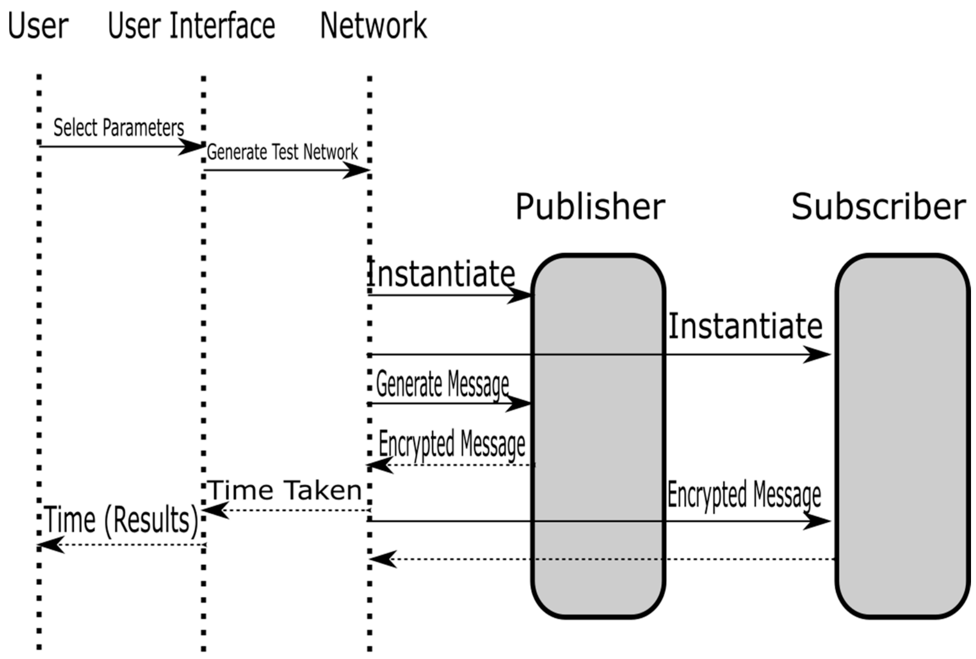

Figure 2. Proposed approach for publisher–subscriber model.

The publisher in the proposed scheme regains an access structure of its subscribers [3]. In this scheme, before a publisher multicasts a message to all its subscribers, the messages are encrypted using one of these algorithms: AES, RSA, and ChaCha20. The use of a particular algorithm depends upon the latency requirements and the security properties we need to maintain. For example, if the scheme offers non-repudiation property, we cannot then use AES, rather RSA would be better to use, as it provides non-repudiation through digital signatures in public-key cryptography. Furthermore, note that using AES does not provide integrity; hence, one needs to use a hash function, such as SHA256, to offer data integrity. We pause this discussion until the next subsection; Section 4 will evaluate their performance to better understand their application and usage in different latency requirements. The publisher encrypts the transmitted message, and multicasts it to all subscribers. Each subscriber decrypts the message and reads the original data. Every exchange of messages contains the ID of the publisher. Algorithm 1 explains the working and various steps of our approach.

Algorithm 1 Proposed Approach

Step 1: A publisher creates a message that is later multicasted in the network for its all subscribers of the topic.
Step 2: An access control list is created based on all subscribers of the topic. A group key is created to cover all subscribers who could receive a secret key to decipher the message.
Step 3: A cipher algorithm is decided to encrypt the message before its transmission over the network. A message is encrypted using a secret key.
Step 4: A secret key is transmitted to all subscribers using a group key, i.e., encrypting the secret key using a group key.
Step 5: The message is reached to all its subscribers who will have access to this message based on the access list.
Step 6: The secret key is extracted using the group key and the message is encrypted by each subscriber.

3.3. Secret Key Exchange

In the case when we use RSA, the public keys of all subscribers and publishers are kept in a public repository, which can be accessed by all authorized publishers and subscribers within the system. We adopted the secret key exchange idea from [3] in the form of an access control list, which is an authorized subset of the subscribers receiving the same message. In the case of AES, the secret key is shared between the publisher and all relevant subscribers. In this case, the symmetric secret key is generated by the publisher each time it publishes data using a pseudo-random number generator. There is a group key 'k' that is shared among a group of publishers and subscribers. This key is used to transmit the symmetric key between the publisher and all its subscribers. This shared secret key is generated each time a published needs to publish a message to its subscribers. Clearly, compromising this key will still not allow an attacker to be able to decrypt the message. We adopted a group key generation from [18], which is based on the logical key hierarchy (LKH) approach. We did not directly use this group key to encrypt a message, as compromising this key may have severe consequences and message confidentiality can be compromised. Instead, we used this key to encrypt the session secret key which is shared between the publisher and all its subscribers. In this case, we use ChaCha20, a stream key that is generated at the publisher, and the same group key is used to send this stream session key to all subscribers of a publisher who wants to publish a message. Each time a message is published, a counter value is also sent along with the message, which reflects how many subscribers are there to decrypt the message. Each time a message is decrypted by a subscriber, its counter value is decreased and a return message with the recent value is sent to the publisher.

3.4. Experimental Design

This subsection discusses the implementation of our approach and the different algorithms that we use, including language-specific modules. Furthermore, we discuss implementing Java crypto modules in the encryption method and justifying their use and limitations.

For our proposed approach, replicating an exact model of a smart grid setup is not required. Therefore, we develop a tool to test encryption algorithms to be evaluated for smart grid applications.

Java. As the implementation language, we use Java in our approach as it offers a wide range of importable modules such as the Java crypto packages for cryptographic operations for AES and ChaCha. Towards this end, the most useful crypto-specific modules of Java in our approach are the cipher and key generator classes.

For the RSA algorithm, which is used more as a comparison tool to show the difference to these quicker encryption methods, the java security module is used for the secure random feature. The *Javax.crypto.cipher* class provides the functionality of a cryptographic cipher. To create a Cipher object, the class calls the cipher's *getInstance*() method and passes the name of the requested transformation to it. A transformation is a string that describes the operation (or set of operations) to be performed. It includes the name of a cryptographic algorithm (e.g., AES), and may be followed by a feedback mode and padding scheme. In this case, it includes the mode type of AES and a padding scheme for extra security.

The cipher is then initialized with the mode (either encrypt or decrypt), the secret key, and an initial vector. The final function of the cipher class used is the final encryption of the plain text using the *doFinal*() function which will encrypt or decrypt a byte from the input, so if you are sending a string, it must be converted into bytes first.

The other crypto modules used are from the key generator class which is a re-usable key generation object. It can only be used for symmetric secret key generation. There are two ways to generate a key; in an algorithm-independent manner and an algorithm-specific manner. The only difference between the two is the initialization of the object.

cipher ci = *Cipher.GetInstance*(*"AES/CBC/PKCS5Padding"*); *ci.init*(*Cipher.ENCRYPT_MODE, skey, ivspec*); *encoded* = *ci.doFinal*(*input*);

Nodes. Two separate classes can be constructed for the publisher–subscribers: the sender and the receiver, as they do not have to fill both roles on the network. However, there is limited functionality to handle peer–peer communication between the IEDS on this test network. When used for modeling a publisher–subscriber model, the sender class acts as the publisher and generates the secret key. The main reason for having separate classes for nodes and not using a superclass is that in the publisher–subscriber, the publisher plays a very different role to the subscriber nodes and should not be treated as a similar type [19]. The senders need to generate Keys for AES and ChaCHa20, and the receiver would need the ability to generate the RSA keys. Both nodes would need to be able to perform, encrypt, or decrypt operations given the secret key and any other algorithm-specific required input.

SpeedTester is the class name of the GUI, which uses an active listener and is an extension of the AWT frame class. It contains all the UI elements and handlers and is used to set up the network when the test is started via the start button. The network class stores all the sent network parameters, generating the timing data from each test. It is also responsible for instantiating the sender and receiver class which acts as our IED nodes in the network architecture. It has a unique setup and runtime for each encryption type.

The Sender Class, also doubling as our publisher in the relevant network architecture, contains methods to instantiate each encryption class and a method for using the encryption class to convert a byte input into an encrypted form. For RSA, it requires the public key to be sent to it for encryption. It also has methods to retrieve the secret key and, if needed, the initialize vector IV.

The Receiver Class uses methods to instantiate each encryption type but uses different constructors, as the secret key should already be known. Therefore, the cipher should be made using this to achieve original plaintext. For RSA, the receiver class contains the key generation for the private-public key pair.

Network Class. For testing the algorithms, a testing class needs to be developed to simulate a test network that the GUI could run to get the results. To facilitate this, a sender object and a receiver object could either work as a simple client–server model for simple peer–peer communication or work as a publisher–subscriber model for the GOOSE and SMV communication types. The network would then act as the data stream maintaining subscriber lists in the case of the publisher–subscriber topology and would forward messages to the designated locations. It would also handle the setup for the algorithms, i.e., the secure transmission of the secret keys for symmetric communication.

Each encryption method has its own class that includes two constructors: one for a new cipher and one to generate an existing cipher based on the secret key and/or initial vector. It also includes the encoding and decoding methods. The full user process from selecting the input parameters to the returning of the timing results can be realized as a sequence process. It details the user and the main classes, not including the encryption classes that the publisher and subscriber employ for the encryption–decryption process. This also highlights the communication flow around the classes. The time will start from the message generation and stop when the subscriber has decrypted the encrypted message for the timing results.

3.5. Implementation: Protecting Communicated Information—Confidentiality

Encryption algorithms can be split into two kinds: asymmetric and symmetric ciphers. Considering the time criticality, we utilize symmetric ciphers; their quicker computation time complies with system requirements. However, the paper includes the RSA example; it is secure and can be used as a baseline comparison for other algorithms. All the encryption algorithms take a byte-input of variable size for encryption and decryption.

Asymmetric Communication RSA. For complete security, RSA is commonly used for secure communication [20]. It is asymmetric encryption with two distinct keys: a public key for encryption and a private key that is used to decrypt. The security is based on the difficulty in factorizing the product of two large prime numbers. The algorithm can be split into major sections: key generation, key sharing, encryption, and decryption.

Key generation for RSA first revolves around selecting two distinct prime numbers p, q, which should be chosen at random and have similar but different lengths to increase the difficulty of factorizing. The next step is computing $n = p{*}q$, n is the modulus for both the public and private keys, its length is the key length. N is released as part of the public key. For the private key, it is the *lcm of* $(p\text{-}1)(q\text{-}1)$, which is calculated as follows.

For encryption and decryption, Java's inbuilt modPow function using the respective public or private keys is applied. RSA has the added benefit of being able to transmit the public key and modulus as part of the public key as plain text without compromising the security of the communication. The encryption example is shown below.

privateKey = *publicKey.modInverse*(*phi*); *BigInteger phi* = (*p.subtract*(*one*).*multiply*(*q.subtract* (*one*)); *BigInteger encrypt*(*BigInteger message*)*return message.modPow*(*publicKey*, *modulus*);

Block Cipher AES. The specifically focused algorithm in this paper is the widely used block cipher advanced encryption standard (AES), which replaced the data encryption standard (DES) [21]. It uses a fixed block size of 128 bits and a key size of 128,192 or 256 bits. AES is a great block cipher to use in the comparison due to its efficient implementation in both software and hardware [22]. The key size that is used in the AES dictates the number of rounds that the plaintext goes through. Step 1 is a key expansion that derives around the key from the cipher key. AES requires a separate 128-bit round key block for each round, along with a final one. In addition, an initial round key is added, which combines (using bitwise XOR) each byte of the states with a byte from the round key. For the next rounds (depending on the key size), run as follows.

- A non-linear substitution step where each byte is replaced with another according to a lookup table.
- A transposition step where the last three rows of the state are shifted a certain number of steps cyclically.
- A linear mixing operation that operates on the columns of the state, combining the four bytes in each column.
- Round key addition where the subkey is combined with the state.

The linear mixing operation is removed in the final round that involves the same steps as the previous rounds but with Step 3. In terms of security, despite a lot of research into AES, there are no practical attacks on AES, with some side-channel attacks on the implementation of AES, which can be prevented by eliminating the relationship between the leaked information and the critical data.

Stream Cipher ChaCha20. Salsa20 and ChaCha20 are a close family of stream ciphers with the ChaCha variant aiming to increase the diffusion per round and reduce the number of rounds needed for security. The Salsa20 encryption function is a long chain of three operations on 32-bit words [23].

Chacha follows the same base design principles as Salsa20 but has an increased diffusion per round, allowing a smaller minimum number of secure rounds for ChaCHa. The extra diffusion does not come at the expense of extra operations as the ChaCha round has 16 additions, 16 xors, and 16 constant distance rotations of 32-bit words, just like a Slasa20 round. The main speed and diffusion differences come from the quarter round where each word is updated twice and the word matrix, where it is built with attacker-controlled input words on the bottom. ChaCha sweeps the matrix through rows in the same order every round.

This example shows ChaCha20, 20 rounds, w used with an input size of 500 bytes using a WLAN which gives a network latency of 2 ms. We can see an example of extra console output of the average time taken for the key generation, which can be an important consideration for results analysis.

4. Results and Evaluation

In this section, we evaluate the performance of the proposed approach and discuss the results from the testing tool. In addition, we evaluate the suitability of our approach and experimental results concerning the implementation into smart grid infrastructure. For this, we refer to the platform used in [3] and implemented these algorithms using Java. To ensure the suitability over a broad spectrum of the proposed approach in substation automation, we test the communication messages of various lengths for all three algorithms; we group them suitably to compare specific points. Moreover, we discuss the suggested implantation for SMV, MMS, and GOOSE based on the findings. We present our results obtained along with their evaluation and attacks analysis in the subsequent subsections.

4.1. Results Observation

Cycles per Bytes (CpB). To improve the timing, we can use the data to work out other aspects of the algorithm's software implementation performance. Due to the wide variety of variables that can affect the time, it is not as reliable as an evidence source. We can use readings to focus on the cycles per byte performance of the algorithm.

The bytes per second is very easy to calculate; it is the size of the packet being sent, in this case, 100, 200, 500, or 1000-bytes. We recorded the time taken in each of these algorithms. We could also split this into the time taken to encrypt and the time taken to decrypt to work out a cycle per byte for each specific process. For ease of simplicity, we used symmetric algorithms; half of the total time was considered to calculate the cycles per byte for either encrypting or decrypting. The cycles per byte are machine-dependent and can be widely affected. Some processors also are optimized to perform specific algorithms at a quicker pace. The tool used in this work runs on a personal machine with a CPU speed of 3.8 GHz for demonstration purposes; the modern PLCs used in smart girds have dual processors with higher CPU speed. In addition, the process manager in task manager during the algorithm run time, the total percentage of the application's processor can be seen. In most cases, this was a single-digit percentage. Furthermore, encryption process cycles per second can be viewed using the percent value to keep track of its performance.

Encryption Duration. Most results labeled under time taken were the time for the plaintext to be encrypted, sent, network delayed, received, and decrypted. Moreover, additional times, such as setup times and key generation duration, were also considered concerning their impact on the results.

AES. The AES encryption times using CBC mode with padding were taken using 100-, 200-, 500- and 1000-byte message sizes and ran on a 1 to 1 connection. The results were taken 10,000 times with a different key generated for each connection, as shown in Figure 3. For the publisher–subscriber message byte ranges of 150–250 bytes, they had a total time of 0.0147 ms for the 200 bytes on the local machine being used.

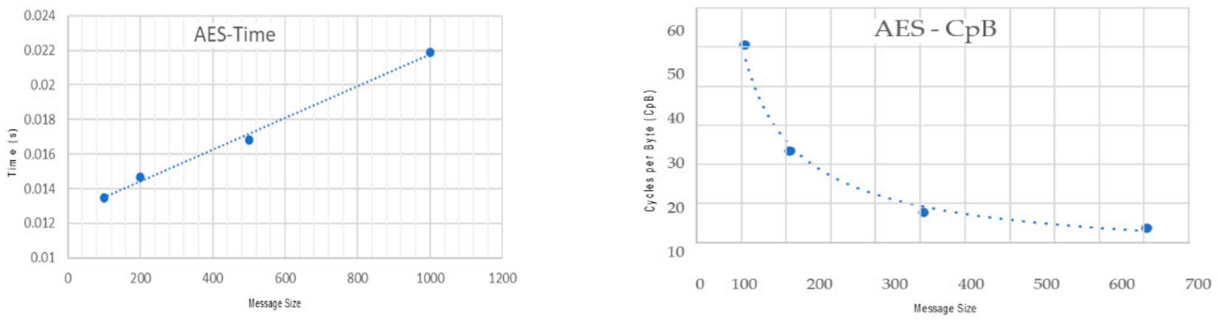

Figure 3. AES Time and Cycle per Byte performance.

According to benchmark studies [24] on a test platform using 2.9e + 009 Hz using CBC mode 128-bit key, which is the same encryption method, AES reached a CpB of 28. The result is comparable to the results of 200-byte input that ran at an estimated 29 cycles per byte 7.8e + 008 HZ. Interestingly, as the byte input increases, the cycles per byte drop and the gradient of the trendline on the AES time results is minimal. Therefore, despite the significant increase in the byte size, it currently suggests a minor increase in operating time. AES is also widely optimized for most current hardware meaning that, in desktops, some AES configurations could still be a faster alternative.

RSA. The RSA used had 2048-bit big integer keys; this is deemed with some of the smaller keys, though quicker, are still deemed to pose a possible security risk. It takes 22 ms to generate the private and public keys, and the public key needs to be sent to the sender.

In a publisher–subscriber model where it is a one-to-many ratio, the publisher would have to store a list of all the public keys of the subscribers subscribed in an offline repository subscriber list [3,19]. Therefore, it takes additional setup time and care and would require more validation for new subscribers. Even on a powerful machine, the run time of RSA is much slower, even with no network latency. The smallest packet size of RSA was 100 bytes and took an average time of just under 12.9 ms. In Figure 4, it can be observed that the RSA could more appropriately be measured in megacycles which means it requires a lot more cycles per byte. To achieve low speeds, it would require an extremely fast cycle speed using optimization not currently available.

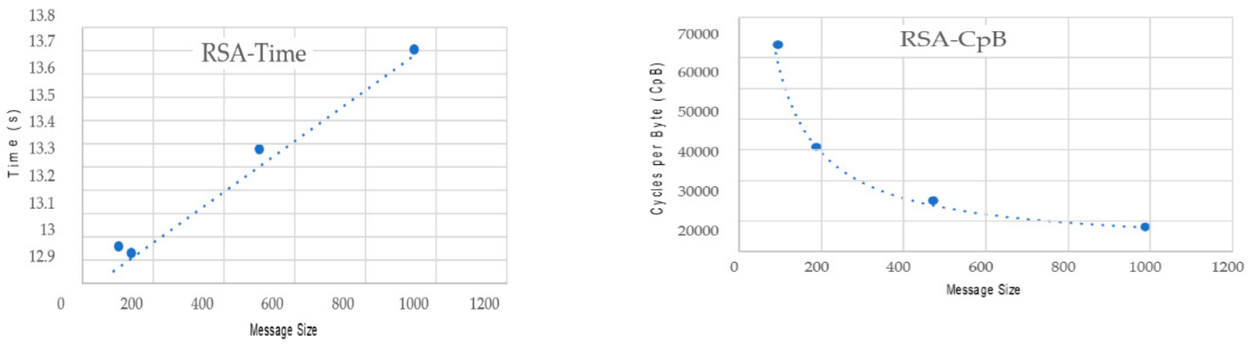

Figure 4. RSA time and cycle per byte performance.

ChaCha20. In Figure 5, the time results for a software implementation of ChaCha20, which belongs to the salsa encryption family, can be seen. ChaCha is the only stream cipher that is tested in this work, and it gives the optimal performance for the application we used. It is tested using the same parameters as the previous algorithms, running each message size 10,000 times for an average result. ChaCha20 had a shallow cycle per byte; the ChaCha20 performs better than other algorithms on lower-performance devices. We can see that cycles per byte do not increase as dramatically as AES or RSA on lower byte inputs. The Salsa20 and ChaCha software speeds in [25] indicate that the cycles per byte depend on the hardware used; the bytes are in the ranges of 3.95 to 15.03. It is also reasonable to suggest lower cycles per byte as more optimizations come in for chach20 as it becomes more widely used.

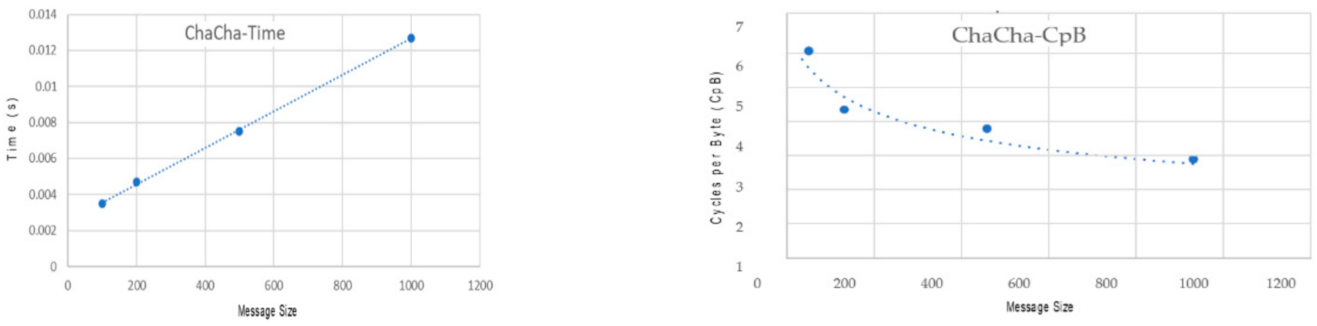

Figure 5. ChaCha20: Time and Cycle per Byte Result.

4.2. Evaluation

Estimated Timings. To increase the usability of the findings, we could use the cycles per byte, calculated at 5.3, and find the CPU speed of processors used in a protection relay to estimate projected times for an IED. Many different processors could be used in various architectures; this work uses the Sitara AM335X processor, which is one of the most popular processors for industrial HMI applications [26]. The aforesaid processor has two different speeds; we used the lower speed of 600 MHZ based on our calculation.

In Figure 6, the estimated times in seconds for both AES and ChaCha20 on the processor using the cycles per byte at each message size can be observed. It is also worth pointing out that ChaCha20 performs better than AES in mobile and lower power devices due to being more lightweight, while, in hardware on optimized machines, AES can outperform ChaCha20.

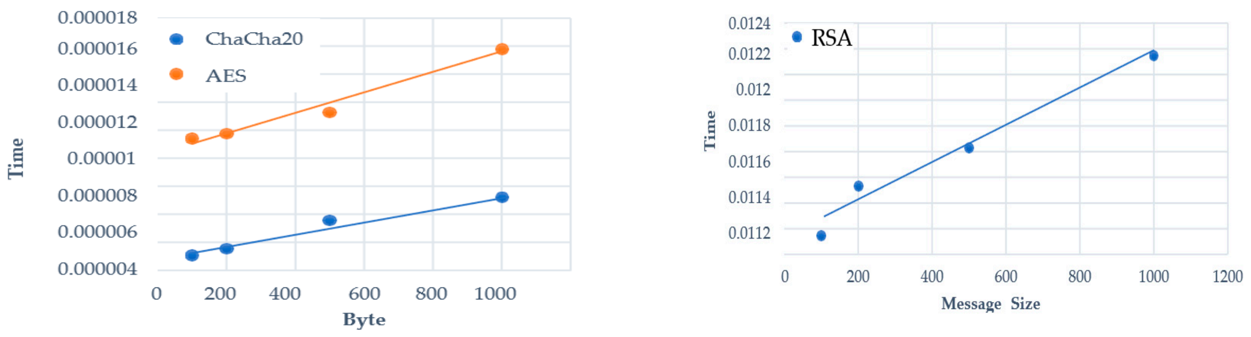

Figure 6. Estimated Time: AES, ChaCha20 and RSA.

GOOSE and SMV. The timing requirements of GOOSE and SMV are 4 ms, and the sizes are 92–250 and 150 bytes, respectively. On average, we can see from Figure 6 that both AES and ChaCha20 can be used even if the network latency is around 2 ms. The network latency of a 4G network means that it will not meet timing requirements. The theoretical limit of 5G is 1 ms; although it would not reach this time in a high connectivity area, it could be possible to have an IED on a 5G connection. RSA has too low of a latency time, as seen from the estimated times RSA graph, to be considered for this connection time and due to the publisher–subscriber architecture would not be a suitable encryption method. Due to ChaCha20 performing better on less powerful machines in an IED scenario, it is a better choice for a smart grid environment and is less common than AES, which had more attention and cryptanalysis for potential attacks. Both algorithms can potentially prevent integrity attacks, such as MITM attacks if the secret keys remain secure.

MMS. MMS has much slower critical times depending on the messages using the protocol; they range from 500 to 1000 ms and work on a more traditional peer–peer architecture. Though the RSA could support this message type, it would require each sender to have the public keys for every node it is sending. It would be simpler to implement one of the other algorithms proposed, such as ChaCha20, and would be easier to set up as all message types would be using the same communication method.

4.3. Attack Analysis

4.3.1. What If the Secret Session Key Is Compromised?

In the proposed approach, it is very unlikely that an adversary can compromise the secret key of the encrypted message by performing an MITM attack. As it will be encrypted using ChaCha20 which generates a stream key every time, the publisher needs to encrypt a message. The key is transmitted to all subscribers using a group key. This means that, even if the attacker has access to the encrypted message, he/she cannot decrypt it successfully. In the worst case, if this key is compromised, it will not impact future messages, as each time a new key is generated for ciphering the message to maintain confidentiality. Hence, data tempering will not work.

4.3.2. What If the Group Key Is Compromised?

In case the group key is compromised, first of all, it does not impact the other messages published under different topics, as each topic is having different subscribers and a different group key. This each group key is shared among a group of a publisher and subscribers, and at the end, a publisher can know the value of its counter, i.e., how many users have decrypted the message. If anyone, not in the access control list, tries to decrypt the message (who did not receive the message authentically), the counter value will not decrease and the return message will send the same counter value. As a result, the publisher will receive the two same counter values; it is good to know that someone not authorized has accessed the message, and a new group key is needed to be created and distributed to all relevant subscribers to that topic. In the worst case, if we assume that an adversary can compromise the group key, he/she may extract the session key (stream) to decrypt a message. However, the adversary cannot decrypt the future messages, as a new group key will be generated and distributed, which will prevent the adversary to extract the correct key.

5. Conclusions and Future Work

In this work, we present a secure, practical, and simple solution to solve security issues in a smart grid; more specifically, we address the security issues of industrial communication protocols such as GOOSE, SMV, and MMS. The key elements of our solution to integrity attacks against the aforementioned protocols include time, performance, and security. Furthermore, a demo tool is developed to investigate the performance of encryption algorithms with various potential factors such as network types and the size of the messages. The experiments reveal that our proposed scheme: (1) can protect the publisher–subscriber model communication against various attacks such as MITM and data tampering attacks; (2) is a practical and simple security solution that complies with the stringent timing requirements in the substation automation systems; and (3) is a lightweight scheme that can be scaled up and employed in larger network setups.

Future work includes (1) a performance evaluation of other encryption schemes over a broader spectrum of cyber-attacks; (2) an exploration into the security solution by considering a trust model where subscriber and publisher cannot be trusted; and (3) an integration of the proposed solution into the IEDS' modules of various applications including merging unit, and then performance will be analyzed in the testbed scenario.

Author Contributions: Conceptualization, N.S. and F.O.; methodology, N.S.; software, F.O.; validation, N.S., F.O. and M.N.N.; formal analysis, N.S., F.O. and B.J.C.; writing—original draft preparation, F.O. and N.S.; writing—review and editing, N.S., M.N.N. and B.J.C.; supervision, N.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Hong, J.; Karnati, R.; Ten, C.W.; Lee, S.; Choi, S. Implementation of Secure Sampled Value (SeSV) Messages in Substation Automation System. *IEEE Trans. Power Deliv.* **2021**. [CrossRef]
2. Markets, R.A. The $39.9 Billion Worldwide Substation Automation Industry Is Expected to Reach $54.2 Billion by 2026. GlobeNewswire News Room. Available online: https://www.globenewswire.com/news-release/2021/06/04/2241918/28124/en/The-39-9-Billion-Worldwide-Substation-Automation-Industry-is-Expected-to-Reach-54-2-Billion-by-2026.html (accessed on 8 July 2021).
3. Saxena, N.; Grijalva, S.; Choi, B.J. Securing restricted publisher-subscriber communications in smart grid substations. In Proceedings of the 10th International Conference on Communication Systems & Networks (COMSNETS), Bangalore, India, 3–7 January 2018; pp. 364–371.
4. Aftab, M.A.; Hussain, S.S.; Ali, I.; Ustun, T.S. IEC 61850 based substation automation system: A survey. *Int. J. Electr. Power Energy Syst.* **2020**, *120*, 106008. [CrossRef]
5. Saxena, N.; Grijalva, S. Dynamic secrets and secret keys based scheme for securing last mile smart grid wireless communication. *IEEE Trans. Ind. Inform.* **2016**, *13*, 1482–1491. [CrossRef]
6. Saxena, N.; Grijalva, S. Efficient signature scheme for delivering authentic control commands in the smart grid. *IEEE Trans. Smart Grid* **2017**, *9*, 4323–4334. [CrossRef]
7. Hussain, S.S.; Farooq, S.M.; Ustun, T.S. A method for achieving confidentiality and integrity in IEC 61850 GOOSE messages. *IEEE Trans. Power Deliv.* **2020**, *35*, 2565–2567. [CrossRef]
8. Elbaset, A.A.; Mohamed, Y.S.; Elghaffar, A.N.A. IEC 61850 Communication Protocol with the Protection and Control Numerical Relays for Optimum Substation Automation System. *J. Eng. Sci. Technol. Rev.* **2020**, *13*, 1–12.
9. Pal, A.; Jolfaei, A.; Kant, K. A Fast Prekeying-Based Integrity Protection for Smart Grid Communications. *IEEE Trans. Ind. Inform.* **2020**, *17*, 5751–5758. [CrossRef]

10. Nweke, L.O.; Weldehawaryat, G.K.; Wolthusen, S.D. Adversary model for attacks against IEC 61850 real-time communication protocols. In Proceedings of the 16th International Conference on the Design of Reliable Communication Networks (DRCN), Milan, Italy, 25–27 March 2020; pp. 1–8.
11. Quincozes, S.E.; Albuquerque, C.; Passos, D.; Mossé, D. A survey on intrusion detection and prevention systems in digital substations. *Comput. Netw.* **2021**, *184*, 107679. [CrossRef]
12. Wannous, K.; Toman, P. IEC 61850 communication based distance protection. In Proceedings of the 15th International Scientific Conference on Electric Power Engineering (EPE), Brno-Bystrc, Czech Republic, 12–14 May 2014; pp. 107–112.
13. Saxena, N.; Choi, B.J. Integrated Distributed Authentication Protocol for Smart Grid Communications. *IEEE Syst. J.* **2018**, *12*, 2545–2556. [CrossRef]
14. Hou, D.; Dolezilek, D. IEC 61850–What It Can and Cannot Offer to Traditional Protection Schemes. 2008. Available online: https://cdn.selinc.com/assets/Literature/Publications/Technical%20Papers/6335_IEC61850_DH-DD_20080912_Web.pdf (accessed on 20 July 2021).
15. Yeh, T.H.; Hsu, S.C.; Chung, C.K.; Lin, M.S. Conformance test for IEDs based on IEC 61850 communication protocol. *J. Power Energy Eng.* **2015**, *3*, 289. [CrossRef]
16. Falk, H. Securing IEC 61850. In Proceedings of the IEEE Power and Energy Society General Meeting-Conversion and Delivery of Electrical Energy in the 21st Century, Clemson, SC, USA, 14–17 March 2006; pp. 1–3.
17. Fateri, S.; Ni, Q.; Taylor, G.A.; Panchadcharam, S.; Pisica, I. Design and analysis of multicast-based publisher/subscriber models over wireless platforms for smart grid Communications. In Proceedings of the 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, Liverpool, UK, 25–27 June 2012; pp. 1617–1623.
18. Kabra, A.; Kumar, S.; Kasbekar, G.S. Efficient, Flexible and Secure Group Key Management Protocol for Dynamic IoT Settings. *EAI Endorsed Trans. Internet Things* **2021**, *7*, 1–17. [CrossRef]
19. Ozansoy, C.R.; Zayegh, A.; Kalam, A. The real-time publisher/subscriber communication model for distributed substation systems. *IEEE Trans. Power Deliv.* **2007**, *22*, 1411–1423. [CrossRef]
20. Galla, L.K.; Koganti, V.S.; Nuthalapati, N. Implementation of RSA. In Proceedings of the International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT), Kumaracoil, India, 16–17 December 2016; pp. 81–87.
21. Rebeiro, C.; Selvakumar, D.; Devi, A. Bitslice implementation of AES. In *International Conference on Cryptology and Network Security*; Springer: Berlin/Heidelberg, Germany, 2006; pp. 203–212.
22. Yuan, Y.; Yang, Y.; Wu, V.; Zhang, X. A high performance encryption system based on AES algorithm with novel hardware implementation, In Proceedings of the 2018 IEEE International Conference on Electron. Devices and Solid State Circuits (EDSSC), Shenzhen, China, 6–8 June 2018; pp. 1–2. [CrossRef]
23. Bernstein, D.J. The Salsa20 Family of Stream Ciphers. In *New Stream Cipher Designs*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 84–97.
24. Cryptopp.com. 2021. Speed Comparison of Popular Crypto Algorithms. 2021. Available online: https://www.cryptopp.com/benchmarks.html (accessed on 6 August 2021).
25. Bernstein, D.J.; ChaCha, a Variant of Salsa. Workshop Record of SASC: The State of the Art of Stream Ciphers. Available online: https://cr.yp.to/chacha/chacha-20080120.pdf (accessed on 12 August 2021).
26. Mundra, A.; Agarwal, M. Human Machine Interface (HMI) for Protection Relay Reference Design; [ebook] Texas. 2017. Available online: https://www.ti.com/lit/ug/tidudn5/tidudn5.pdf?ts=1627122810724&ref_url=https%253A%252F%252Fwww.google.com%252F (accessed on 26 August 2021).