A Fast Prekeying Based Integrity Protection for Smart Grid Communications

Amitangshu Pal*, Alireza Jolfaei†, Krishna Kant*, Haotian Chi*
* Temple University, Philadelphia, PA
† Macquarie University, Australia
E-mail:{amitangshu.pal@temple.edu, alireza.jolfaei@mq.edu.au, kkant@temple.edu, haotian.chi@temple.edu}

***Abstract*—Due to the mission-critical nature of energy management, smart electric grid is a prime target for cyber-attacks. Maintaining the integrity of smart grid communications is thus crucial to avoid harmful actions triggered by data corruption or false data injection. Because of the very tight latency requirements for some of these communications, the standard integrity mechanisms are too expensive and such communications are often left unprotected. In this paper, we propose a prekeying based integrity protection (PreKIP) mechanism that computes the key for the next message in advance followed by a simple exclusive-or operation with the message when it is generated. This allows for very fast encryption that protects both integrity and confidentiality of the messages. The rigorous security analysis shows that the proposed method is secure against CRC and message replay attacks. A comparison with existing integrity protection methods showed that the proposed method is much faster (upto 21 times faster than the HMAC scheme, and even faster as compared to other crypto algorithms) and it meets the strict 3 ms timing requirement in power protection applications.**

Keywords*—*Smart Grid, integrity protection, GOOSE protocol, Sampled Value protocol, IEEE C37.188

I. INTRODUCTION

The emerging smart grid architecture uses real-time monitoring and control of the power grid in order to provide high efficiency, stability and robustness in the power supply. A series of communications protocols have been defined for this purpose over the years. In particular, there are 3 protocols in existence currently. The earliest one is the IEEE C37.118, which is now split into two parts, with C37.118.2 being aligned with more widely deployed IEC 61850-9-5 for synchrophaser communications [1]. It is currently the most widely deployed protocol for synchrophasors but lacks security. On the other end is the DoE developed STTP protocol [1] which has been recently picked up by IEEE for standardization but currently not deployed except on an experimental basis. Therefore, we will largely focus on IEC 61850-9-5 which is rapidly being adopted world-wide for deployment.

Although IEC 61850-9-5 includes integrity protection mechanisms for the communications; they are optional for critical messages requiring very low latency. Without any integrity protection, active attackers may intercept, modify, inject or replay messages to launch various attacks. As an example of possible attack, if a protection message is falsely set to indicate abnormal voltage or current value, it could trigger protective relays and/or the generation control equipment to react, potentially leading to blackouts. It is worth mentioning that even for less latency sensitive applications, the penetration of elaborate IEC 61850-90-5 security procedures into real implementations is likely take many more years.

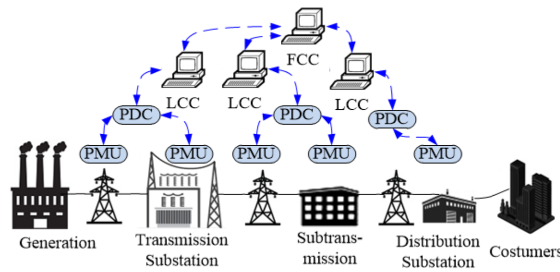


Fig. 1. Typical model of a power grid architecture.

In this paper, we propose a lightweight prekeying based integrity protection mechanism, named PreKIP, for carrying such messages. The key idea in our approach is to generate the keys dynamically based on a salting procedure independently on both send and receive sides and have it ready for use by the next message. The "use" is merely a very fast XOR operation and thus can achieve extremely low latency. The rigorous security analysis shows that the method can successfully thwart both ciphertext-only attacks and known/chosen plaintext attacks. A comparison with existing integrity protection methods shows that despite having the same level of computational complexity, the proposed method is upto 21 times faster than the HMAC scheme (and even faster than others), and it is the only integrity protection scheme that meets the strict timing requirement, that is, 3ms.

The remainder of this paper is organized as follows. Section II discusses the background of smart grid communications including protocols and the need for integrity protection. Section III discusses our proposed method and Section IV provides the security analysis. Section V evaluates the computational complexity and running time of the proposed algorithm. Section VI summarizes the related work. Finally, the paper is concluded in section VII.

II. SMART GRID ARCHITECTURE AND COMMUNICATIONS

A. Smart-Grid Architecture

The emerging smart grid architecture uses PMUs to continuously monitor line data (for example, voltage, phase, frequency, and GPS location) and communicates them to the supervisory control and data acquisition (SCADA) systems to ensure that any issues related to grid health are handled promptly. Fig. 1 shows the overall architecture where the data from PMUs is "concentrated" through Phasor data concentrarors (PDCs) installed in key substations. PMUs collect and send samples 30 or 60 times a second through a publish-subscribe mechanism, where the PMUs work as publishers to which the PDCs subscribe. A PDC receives data from many (typically 3 to 32) PMUs, and then sorts and aggregates the received data based on the time-tag. The aggregated data is then relayed using a two-way communication system to a number of local control centers (LCCs), which coordinate their actions interacting with a federated control center (FCC). Subsequently, LCCs draw the best overall snapshot solution using all PMU measurements [2]. Control centers use the IEC 61850-90-5 standard to communicate with smart measurement units [3].

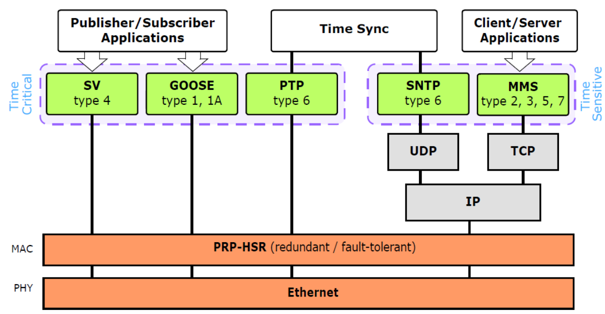


Fig. 2. An Illustration of Smart Grid Communications Protocols

The IEC 61850-9-5 standard is itself a collection of protocols, each defined for a different set of smart grid applications. This is shown in Fig. 2. The MMS (Manufacturing Message Specification) protocol connects communications centers and gateways through a client-server connection with the IEDs (Intelligent Electronic Devices) inside the substation and TimeSync provides time synchronization. The two important protocols of interest here are SV or SMV (sampled value) and GOOSE (generic object oriented station event), and both were originally defined to operate directly on top of Ethernet and thus are not routable. SV protocol is used to exchange messages containing samples of electrical quantities such as the very fast rate sampling of voltage/current at a relay. While SV was originally designed for intra-substation use, it is also useful for sending the PMU data stream to PDCs and LCCs. The GOOSE protocol is used to exchange information between IEDs (Intelligent Electronic Devices) based on the important events that occur in the system. Such messages include power measurements going between protection relays, status updates, sending command requests, and critical messages that demand immediate action (e.g., relay trip). The use of GOOSE for transporting PMU data is also reported in several publications [4], but in this case, repetition of the message would not be useful. Both GOOSE/SV messages require very low transmission latency, which is the main reason for running them directly on layer-2. However, more recently the routable version of these protocols known, respectively, as R-SV and R-GOOSE have been defined, so that it is possible to transport data from several substation networks to a concentrator or control center over UDP or TCP. Tunneling can also be used to carry layer-2 GOOSE/SV messages to points outside a substation.

TABLE I. PAYLOAD DATA SIZE UNDER VARIOUS PMU/PDC CONFIGURATIONS

Number of PMU voltage channels	PMU data size (bytes)	PDC data size			Applicability
		3 PMUs	6 PMUs	10 PMUs	
1	40	104	200	328	Single-phase lines in distribution systems
3	56	120	216	344	3-phase power transmission lines
6	80	144	240	368	Power transmission lines
8	96	160	256	384	Power transmission lines
10	112	176	272	400	Power transmission lines

Assuming that the PDC's are located on the local substation network, SV/GOOSE can be used for transmission from PMU to PDC. However, since LCC is must necessarily be located remotely and connect to multiple substations, all exchanges between PDCs and LCC must be using either a routable protocol such as R-SV/R-GOOSE or by tunneling the SV/GOOSE protocols. In the following, we largely focus on non-routable versions of SV/GOOSE which can be carried over Ethernet frames riding a SONET or similar lower-level network. These are appropriate for low-latency, and low-packet-loss communications. R-SV/R-GOOSE instead usually ride TCP and are more appropriate for less critical communications. Table I gives the size of payload data in transmission and distribution substations under various PMU/PDC configurations.

B. GOOSE Protocol

Both GOOSE and SV protocols operate on publish-subscribe principle. That is, each IED publishes its data, and other entities interested in the data (IEDs, PDCs, LCCs) subscribe to the data stream. In such an environment, the communication is largely one-way; the subscriber does not send any ACK/NACK to the publisher and thus the publisher cannot tell if the data was received correctly. The publisher can, however, retransmit the data.

When GOOSE is used for communicating events, it will typically generate bursts of messages (e.g., on an over-voltage detection), with significant quiet periods in between. It continues to retransmit these events (with successively increasing gap between retransmissions) until a new event occurs. In such situation, the transmission interval exponentially increases to the normal periodic interval as shown in Fig. 4. Assume that T_0 is the time interval between GOOSE messages in periodic mode, and $T_1, T_2, \ldots, T_n = T_0$ are the time intervals in the burst mode, with $T_1 < T_2 \ldots < T_n$. In IEC 61850, T_0 is typically in the range of 5–100 ms, whereas T_1 is in the range of 0.5–5 ms [5]. After the first retransmission after T_1, each successive intervals doubles until it reaches T_0. The requirements for the delivery of GOOSE messages are pretty stringent; the messages should be delivered within 4 ms from the time an event occurs to the time the message is received for protection and control applications; this requirement is revised to 3 ms in IEC 61850-5 [6].

Fig. 3(a) shows the composition of a GOOSE PDU, named `APDU` [7]. GOOSE tries to deliver messages in sequence by using two fields called `stNum` (state number) and `sqNum` (sequence number). Every time a new event occurs, the transmitter increments `stNum` and transmits a

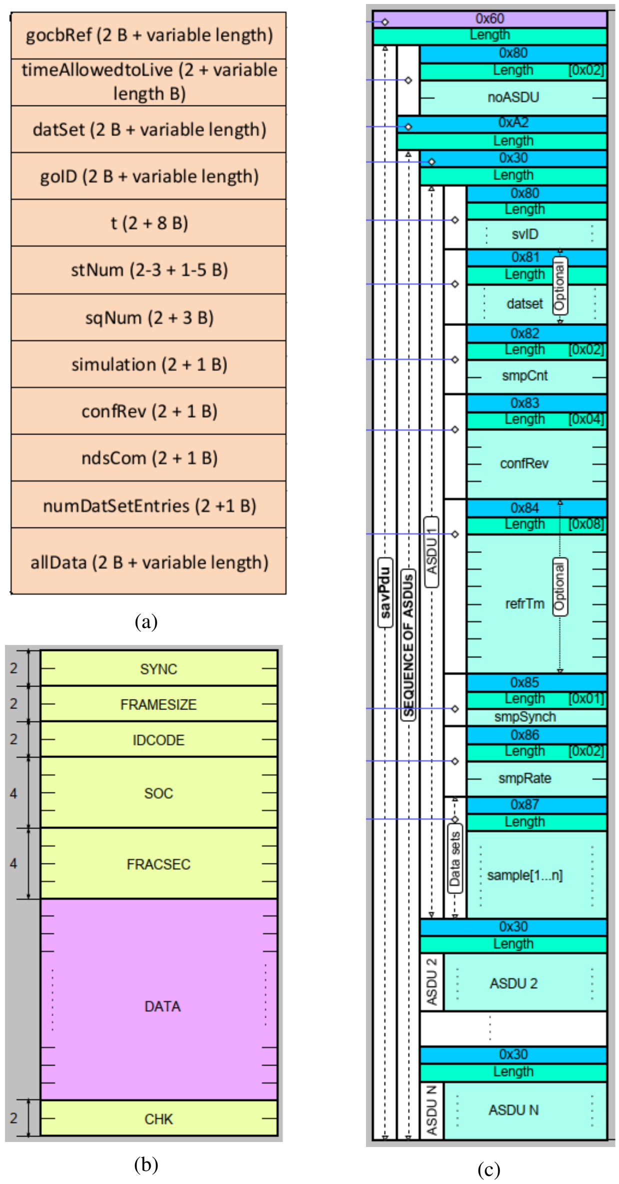


Fig. 3. Format of (a) GOOSE, (b) IEEE C37.188.2 and (c) SV messages.

new message with this `stNum`. If no event occurs and a time `timeAllowedtoLive` elapses, the transmitted simply repeats the last message. Note that if the next event happens rather quickly, it is considered to override the previous one and hence no repetition of previous message is done. The counter `sqNum` is incremented each time the previous message is repeated.

C. Sample Value Protocol

As stated earlier, SV protocol is primarily intended for fast transmission of samples of analog measurements from various substation devices on to the local bus for which it may use very high sampling rates (e.g., 80 to 256 samples per cycle). However, for use in the PMU context, the rates will usually be far lower, perhaps less than once per cycle. Like GOOSE, it also uses the publisher/subscriber model, it does not retransmit data, since retransmission is not meaningful for data stream delivery. Each SV PDU can include several measurements, one per ASDU (application service data unit) as shown in Fig. 3(c). Each ASDU contains the measurement data (e.g., voltages and currents for each phase in a 3-phase circuit) plus some predefined fields, only one of which is of interest here. This is `smpCnt`, which is incremented each time a new data sample in ASDU is taken. `smpCnt` is only 2-bytes and it is reset every second through the TimeSync based synchronization protocol.

The sampling rate of SV involves two factors: measured signal frequency and Samples Per Periode (SPP). IEC 61850-9-2LE defines two SPP values of 80 and 256. Thus, if the measured signal frequency is 50 Hz and SPP is 80, then the sending time interval is 1/50/80, or 250 μs [9]. Fig. 5 shows the packet transmission frequencies of different formats proposed in IEC 61869-9 which is based on IEC 61850 series [10]. Fig. 5 shows that the packet transmission frequencies vary from 2400–5760 Hz, i.e. the transmission intervals vary from 173–416 μs. To ensure that the messages are not delayed, the samples need to be packetized before the next sampling time (i.e. much less than 250 μs). Similar to GOOSE messaging, the SV messages are also time sensitive the, thus messages should be delivered within 3 ms.

D. IEEE C37.118.2 Protocol

Even though this protocol is being replaced by IEC 81650-9-5, it remains the most widely deployed protocol for PMU communications and also lacks security. Each synchrophasor measurement is tagged with a UTC timestamp consisting of three components: 1. Second-Of-Century (`SOC`), 2. FRACtion-of-SECond (`FRACSEC`), and 3. Message time quality flag, which is shown in Fig. 3(b). The `SOC` count is a 4B integer count in seconds from UTC midnight (00:00:00) on January 1, 1970. Each second is divided into an integer number of subdivisions by the TIME BASE parameter that is defined in configuration frame. The `FRACSEC` count is an integer representing the numerator of the `FRACSEC` with `TIME BASE` as the denominator. The time quality flag provides the accuracy of the time measurement.

E. Protecting Messages

In general, the adversaries may be *passive* or *active*. Passive attackers eavesdrop on the communications between devices by wiretapping to the links between end devices and the substation switches to obtain the message contents and traffic characteristics of the substation; they neither modify the messages in the channel nor communicate with the end devices. The objective of active attacks includes learning more about substation's operations which can be helpful in later disruptions. In fact, with the dramatic rise in analytics capabilities, a long-term silent monitoring could derive important information about the types and sources of messages. For example, an attacker may be able to determine what type of perturbation will cause the greatest disruption in the power flows through active attacks. Thus, encryption may be desirable for privacy and integrity purposes, although ensuring communication availability is regarded as the primary goal in smart grids.

However, the current smart grid deployments generally do not use any encryption or integrity protection to reduce

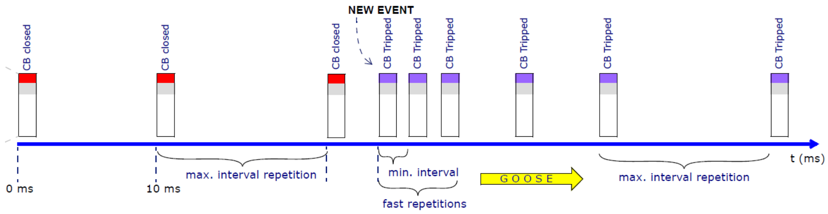


Fig. 4. Timing diagram of GOOSE message transfer.

Sampling Frequency	Samples/ packet	Packet Frequency
4000 Hz (80 SPC @ 50 Hz)	1	4000 Hz
4800 Hz (80 SPC @ 60 Hz)	1	4800 Hz
12800 Hz (256 SPC @ 50 Hz)	8	1600 Hz
153600 Hz (256 SPC @ 60 Hz)	8	1920 Hz
4800	2	2400 Hz
14400	6	2400 Hz
5760	1	5760 Hz

Fig. 5. Packet Frequencies of SV in IEC 61869-9 [8].

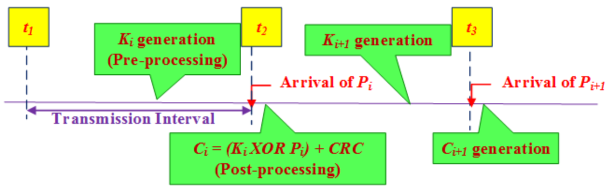


Fig. 6. Timing diagram of PreKIP.

communication latency in time-critical applications; for example, the standard for formatting and delivery of PMU data (IEEE Standard C37.118 [11]) includes no end-to-end security mechanisms. Although a standard IEC 62351 introduces several message authentication code (MAC) algorithms to protect GOOSE integrity, but it still allows an escape route for latency critical messages by specifying an identifier in the message header to zero. Thus for time critical messages requiring a latency of 3 ms, no encryption is likely to be used. This includes GOOSE and SV messages that are used to transmit both the streaming PMU data and the event based data concerning load shed [12] and synchrophasor-assisted transfer trip [13]. The latter involves sending a trip signal from one substation to another that could be more than 100 miles away.

We make no assumption on the attacker's ability to disrupt communications by intruding the system via physical or network based access to the substation switches. In this paper, we focus on hardening the communication protocol and ensuring the end-to-end confidentiality and integrity with assumption of trusted and uncompromised IEDs. Attacks that exploit the hardware or firmware vulnerabilities to bypass the integrity protection are beyond the scope of this paper and should be taken care of by the device manufacturers and substation operators to guarantee the authenticity of original data being protected by our scheme.

Because of the one-way communication and a quiet overriding of previous message with a new one in GOOSE/SV, it is simply not possible to synchronize the two sides precisely or account for every message. Note that the PDC does have the capability to send a message to PMU for special purposes (e.g., key exchange) which are outside the scope of the proposed mechanism. In this paper, we develop a scheme of having the key for the next message to ready so that the next message can be encrypted via a simple XOR operation. Thus, so long as there is some minimum gap between the successive messages, the additional latency caused by the encryption can be kept almost negligible. Unfortunately, the lack of synchronization makes this a difficult proposition in general, and we take advantage of the special context of various message types.

III. Proposed Integrity Protection Scheme

The purpose of the proposed mechanism is to ensure integrity via the use of CRC and confidentiality via a fast encryption mechanism that generates a new key for almost every message between transmissions so that the latency can be largely hidden. This is summarized in Fig. 6. As depicted in Fig. 6, the key K_i corresponding to P_i is calculated before it arrives; thus, after the packet arrives, the payload is XOR-ed with the key to produce the ciphertext C_i before transmitting. The XOR operation is very fast and does not have any significantly contribution to the encryption latency. Unfortunately, the use of XoR along with CRC introduces some vulnerabilities that we address by obfuscating the CRC, as discussed below.

A. Key Stream Generation

Assume that the lengths of the original message and the CRC checksum be L_m and L_c bits respectively. To this, we add an additional $2L_c$ bits for CRC obfuscation (see Section III-E), and thus get a plaintext of length $L = L_m + 3L_c$ bits. Now, both the transmitter and receiver need to generate the same key stream KS of length L bits for a message without any explicit handshake. In the following we discuss the generation of KS, which will be ultimately XOR'ed with the plaintext to generate the ciphertext.

To generate KS for the jth new message, we start with a one-time key sk_j which is defined as

$$sk_j = \mathcal{K} || salt_j, \quad (1)$$

where $||$ refers to the concatenation of a pre-distributed key $\mathcal{K}$ and a per-transmission "salt" to make the key unique to each transmission. We assume that $\mathcal{K}$ is either manually configured on both transmit and receive side, or communicated in an out-of-band manner. Note that once we establish an encrypted communication channel, $\mathcal{K}$ can be changed easily through a handshake, and the best practice demands that this be done periodically (e.g., once a month). The length of the preconfigured key $\mathcal{K}$ should be adequate to avoid brute-force attacks, e.g., 128 bits.

However, $salt_j$ needs to be generated correctly for each message by both transmit and receive sides without any further message exchanges, as we shall discuss shortly in Subsections III-B to III-D for different communication protocols. The resultant one-time key sk_j is not used directly for encryption; instead, we compute a one-way, secure hash H over it so that the KS does not reveal the key. The hash H could be SHA1, SHA256 or another suitable one-way function. Notice that the resulting message digest, say d_j, will be a fixed length value L_d (e.g., 160 bits for SHA1 or 256 bits for SHA256).

$$d_j = H(\mathcal{K}||salt_j) \tag{2}$$

Thus to generate a long enough key stream, we utilize the *avalanche effect* of the hash function H to output significantly changing digests by adding an additional counter into the salting. By concatenating the counter's value cnt from 0 to $N-1$ to $salt_j$, we obtain N salts for each message, which are used to generate N hash digests $d_j^{(i)}$, $i = 1..N$. The final key stream is then a concatenation of the hash digests, i.e.,

$$KS = d_j^{(1)}||d_j^{(2)}||\cdots||d_j^{(N)}, \tag{3}$$

where N is chosen as $N = \lceil \frac{L}{L_d} \rceil$ so that the KS is at least as long as L. The KS is then XoR'ed with the plaintext of length L. Our salting scheme differs based on the communication protocols which we discuss in the following subsections.

B. Salting Scheme for GOOSE

Here we consider general GOOSE messages that are initiated by specific events, as well as the retransmitted ones. The *intent* of the salting scheme is to generate a unique KS for every message – including both new and retransmitted messages; unfortunately, given the lack of acknowledgements (ACK or NACK), it is very difficult to realize this intent. As stated earlier, `stNum` is incremented by 1 when the message transmits a new event and `sqNum` is incremented by 1 if the message retransmits an old event. Thus, to realize the intent, we need to set $salt = \texttt{stNum}||\texttt{sqNum}$ for every message, which means that `stNum` and `sqNum` must be transmitted in the clear to receiver, so that it too can construct KS. However, that will expose the keys to the attacker, thus, the receiver must be able to predict the pair $(\texttt{stNum}, \texttt{sqNum})$ *before* it decodes the message, which we discuss in section III-C.

One other situation to consider is the rollover of `stNum`. First, the rollover of 32 bit `sqNum` is a nonissue, since that can only happen if the same message is continues to be retransmitted without success for a very long time. (Note that `sqNum` will be reset to 0 each time a transmission succeeds or the next overriding event occurs.) Furthermore the rollover of `stNum` is easily handled by the modulo arithmetic on both sender and receiver sides, since there is no chance that a rollover would collide with a previous message with $\texttt{stNum} = 0$.

C. Key Synchronization

To decrypt and verify the received message, the subscribers need to use the same key stream KS. To this end, the subscribers need to use the same salts as those on the publisher's side. Recall that each salt is computed from 3 parameters, i.e. `stNum`, `sqNum` and `cnt`. The `cnt` is easy to synchronize since it takes the same values from 0 to $N-1$ for all messages.

The challenge for synchronizing `stNum` and `sqNum` in key stream generation is that both a publisher and its subscribers cannot predict the next message is to transmit a new event or to re-transmit an old event (recall that key streams are generated ahead of time and that the publisher keeps retransmitting messages for an event until a new event comes). From the publisher's side, if a new event happens then it needs to send a packet *instantly* with $\texttt{stNum} = \texttt{stNum}+1$ and $\texttt{sqNum} = 0$. As in a pathological scenario multiple events can occur one after other (or within a interval of very short time), the publisher needs to be proactive in making the key streams for the next possible set of event driven messages, with increasing `stNum` values. The publisher can do so by proactively maintain a certain number (say $\mathcal{N}$) of key streams and store it in a queue. Whenever a new event occurs, the publisher can dequeue a new key stream, execute the XOR operation and transmit. This is required only for critical event reporting and not for PMUs. Furthermore, generation of too many critical events at the same time is very unlikely, therefore a rather small value of $\mathcal{N}$ should be enough. The publishers can generate the key streams for the retransmitted messages in between the message transmission intervals, with $\texttt{sqNum} = \texttt{sqNum} + 1$. As the minimum time interval for fast retransmission is around 0.5–5 ms, such key stream generation process needs to be fast enough. Algorithm 1 summarizes the key stream generation for the publisher.

Algorithm 1: Key stream generation at the publisher

```
Input : L ← the required key stream length;
        msg ← last sent GOOSE message;
        K ← secret key;
Output: Key stream KS;
cnt ← 0;
stNum ← (msg.stNum == 2^32 − 1)?1 : msg.stNum;
if msg.isNewEvent() then
  | sqNum ← 0;
else
  | sqNum ← (sqNum + 1 ≤ 2^32 − 1)?(sqNum + 1) : 1;
while cnt < ⌈L/L_d⌉ do
  | salt_cnt ← (stNum||sqNum||cnt);
  | KS ← H(sk_i||salt_cnt);
  | cnt = cnt + 1;
```

On the other hand, the subscriber side also needs to generate the same key stream to decrypt the incoming messages. To ensure this, the subscriber follows the following set of steps after receiving the message. We assume that the subscriber is equipped with a fast computing unit, so it can generate the key streams very fast, and thus can try out multiple key streams for decryption. First assume that the last correctly delivered message was not a retransmission. So the receiver knows the received `stNum`, and $\texttt{sqNum} = 0$. There are 5 cases in this regard, which are followed by the subscriber in sequence if the integrity check fails:

(1) Next message also makes it to the receiver without corruption. In this case, the receiver can predict $(\texttt{stNum} + 1, 0)$, decrypt the message, and do the integrity check. This is the normal case and likely to occur in all but a small number of cases.

(2) Next message is lost but a new message is generated before time for its retransmission and makes it to the receiver. In this case, the receiver can predict $(\texttt{stNum} + 2, 0)$.

(3) Next $k-1$ messages are lost (for some $k > 1$), but the kth message, which is correctly received, is still generated before the retransmission time. In this case, the receiver can still determine how many messages might have been lost and thus predicts $(\texttt{stNum}+k, 0)$ for a suitable k. The probability of

this should vanish very quickly with k. Notice that cases (1)-(3) can be distinguished by the receiver from the retransmission cases by monitoring the time between receives, which in cases (1)-(3) will be smaller than the minimum retransmission time.

(4) No new messages are generated until it is time for retransmission. This is the normal case of retransmission, so the receiver can predict the key stream with $\texttt{sqNum} = \texttt{sqNum}+1$.

(5) Several messages are lost, which include some new messages as well as some retransmitted ones. In such scenario, the receiver tries all combinations of $(\texttt{stNum}+k_1, \texttt{sqNum}+k_2)$, where (k_1, k_2) varies from 0 to a certain threshold $\mathbb{K}$. However, with tunneled GOOSE, the probability of multiple message losses should be negligible.

The most time consuming operation in the key generation is the secure hash; therefore, to ensure that the key is ready for each transmission, we require that the key-stream generation time T_{KS} should satisfy the inequality

$$T_{KS} = N \cdot T_H = \left\lceil \frac{L}{L_d} \right\rceil \cdot T_H \leq T_1. \qquad (4)$$

where T_1 is the minimum retransmission interval and T_H is the time to execute the hashing once.

Note that in case of message loss, the receiver may need to try decoding the message with multiple potential keys. This is acceptable for the following reasons: (a) the receiver, being a PDC or LCC, has substantially more computing power than the sender (a PMU) – for example, a desktop/server level machine as opposed to a micro-controller, (b) the message loss probability is expected to be quite low for non-routable critical communications, and thus trying with multiple keys is needed only occasionally, and (c) when a message loss does occur, the 3 ms latency objective is unlikely to be met already, and a small additional delay should not be significant.

D. Salting Scheme for SV and IEEE C37.118.2

Since SV is specifically intended for streaming data without any retransmissions, the proposed mechanism is ideally suited for it. In particular, assuming a maximum packet transmission frequencies vary from 2400 to 5760 Hz, we have a total of 173–416 μs between samples to generate the key for the next sample. Of course, because of the time needed for other operations, not the entire 173 to 416 μs is available for key generation, but generation times in the range of few tens of μs should be workable.

Now let us consider the salting scheme for SV, which requires a unique sequence number for each sample. We can use the `smpCnt` of the first ASDU as the salt, however, `smpCnt` is only 2-bytes and it is reset every second. Thus, `smpCnt` by itself is unable to provide a unique sequence number for a ASDU. However, this issue can be addressed by defining a virtual sequence number `VsqNum` which is incremented each time the smpCnt of first ASDU is reset to zero. Note that for this to work, we need to make the implicit assumption that the first ASDU sent by a PMU always concerns the same entity (e.g., the same bus). A 32bit counter is quite adequate since it will overflow in about 138 years. Now the pair `(smpCnt, VsqNum)` together can be as the unique sequence number.

TABLE II. KEYED OBFUSCATION AND DE-OBFUSCATION

Key Bits	Obfuscation Operation	De-obfuscation Operation
00	Do nothing	Do nothing
01	Flip the current CRC bit	Flip the current CRC bit
10	Flip all CRC bits except the current bit	Flip all CRC bits except the current bit
11	Rotate the current CRC byte	Reverse-rotate the current CRC byte

IEEE C37.118.2 messages are transmitted with a relatively lower frequencies, around 30–60 frames/seconds [4], which gives a gap of 16.67–33.33 ms. Thus the key generation key of around 10 ms should be sufficient for IEEE C37.118.2. As for IEEE C37.118.2, we can use a combination of `SOC` and `FRACSEC` as salt. The "TIME BASE" that defines the range of `FRACSEC` is a configuration parameter and should not be changed during operation (it may be changed by taking the IED offline and making configuration changes). As the IEEE C37.118.2 messages are generated and transmitted periodically, the receiver can predict the next salt (or set of salts in case of packet loss) for generating the key stream. Effectively, the pair `(SOC, FRACSEC)` can act like a unique sequence number.

E. CRC Obfuscation

The common practice of appending CRC code to the message introduces a vulnerability in XOR based encryption due to the linearity property of CRC [14]. It is easy to verify that CRC($X \oplus Y$) = CRC(X) $\oplus$ CRC(Y). Thus, an attacker could XOR the ciphertext X with an arbitary message Y through a Man-in-the-Middle (MitM) attack and XOR CRC bits with CRC(Y). This is easy to do if the CRC bits appear in a known position in the message (e.g., at the end). The attacker could also choose Y such that CRC(Y) $= 0$. To address this, we employ a keyed obfuscation algorithm to shield the CRC bits, disabling CRC cracking attacks. The obfuscation function should alter the CRC significantly and yet should be easily recoverable using the key. Here we propose a method that not only changes the CRC bits but also introduces uncertainty by using 2 key stream bits for each CRC bit.

The detailed operations are shown in Table II. Without the key for obfuscation, the bits in the CRC segment become obscure and "non-linear" to attackers. It also becomes impractical to perform brute-force attacks since frequent CRC verification failures on the publisher side can trigger alarms, which can be easily achieved by adding this type of anomaly to the alarm systems in current smart grid deployments.

F. Message Embedding and Verification

After obfuscating the CRC, the publisher uses the first $L = L_m + L_c$ bits of KS to encrypt the concatenation of the message and the obfuscated CRC by performing an XOR operation. Upon receiving the message, the subscribers reverse the above process by: (1) decrypting the message with the first $L = L_m + L_c$ of KS; (2) de-obfuscation the CRC based on the operations in Table II; (3) verifying the derived CRC. The received message is untampered if the CRC checking is passed.

Notice that this mechanism achieves the confidentiality using the one-time key encryption mechanism, whereas the message integrity is provided in two ways. First, the receiver predicts the salt for the next message to decrypt it, and after the decryption it matches the salt with the corresponding portion of the message. For example, in case of GOOSE the salt consists of `stNum` and `sqNum`, which can be matched with the original message after decryption. Similarly in case of SV, the `smpCnt` is used in the salt, which can also be checked with that of the decrypted message. In addition to matching the salt fields, CRC can further check for integrity.

G. Secret Key Management

Recall that the initial secret key $\mathcal{K}$ is shared among the publisher and its subscribers. In XOR-based stream ciphers, attackers who have access to the original plaintext afterwards (from published PMU datasets) can gain access to the key stream by XOR-ing the plaintext and the sniffed ciphertext, namely the N hash digests d_j. However, because of the one-way hash function, it is very difficult to derive the original string from there (required to get at the underlying fixed key). Nevertheless, there is some risk in using the same fixed secret key for extended periods. Therefore, the transmitter can exploit our encryption mechanism to occasionally provide a new key to the subscribers.

IV. Security Analysis

The proposed scheme provides confidentiality and integrity in the presence of passive or active attacks. A powerful attacker (for example, an inside attacker) can acquire both the plaintext and ciphertext of messages transmitted in the substation system. Thus, he can obtain the historical key streams easily by performing an XOR calculation over the plaintext and ciphertext of the same messages. Note that the key streams are the digests output by hash functions with the shared secret key and the synchronized salts. Through the historical key streams, the attacker can collect hash samples, mappings between hash salts, and hash digests. Then, the attacker can make attempts to find out the secret keys sk_i with these samples. With a 128 bit preconfigured key, a brute-force attack needs 2^{127} attempts to discover it. The SHA1 secure hash used as function $H()$ is known to have a collision attack length of 63 bits, whereas use of SHA-256 raises it to 128 bits [15], [16]. Both of these are more than adequate particularly since collision attacks are unstructured. In the following, we analyze the security of the proposed method with respect to CRC attacks and replay attacks.

A. CRC Attacks

We assume an active, strong adversary who has access to the encryption machine without the knowledge of the secret key. The adversary can control the parameters `stNum`, `sqNum`, and `cnt`, and can input arbitrary measurements to the encryption machine and generate corresponding ciphertexts. Such an adversary is able to encrypt different measurement payloads using the same key stream. In particular, to mount a successful attack and perturb a message y without disturbing the integrity, the adversary only needs to generate a message x such that CRC(x) = 0. If CRC(x) = 0, then for all y, CRC($x \oplus y$) = CRC(x) $\oplus$ CRC(y) = CRC(y). The adversary can compute $(KS \oplus (x||\text{CRC}(x))) \oplus (KS \oplus (y||\text{CRC}(y))) = (x||\text{CRC}(x))) \oplus (y||\text{CRC}(y))$, where KS is the key stream. If CRC(x) = 0, then for all y, CRC($x \oplus y$) = CRC(x) $\oplus$ CRC(y) = CRC(y).

The CRC obfuscation avoids such an attack by altering the bit value or the position of every CRC bit to the padding key. The padded result is an obfuscated CRC with length more than 32 bits. Without the padding key, attackers cannot determine which bits belong to the original CRC and therefore cannot leverage the linear properties of CRC to perform active attacks. Moreover, the intercepted messages are encrypted with a stream cipher, which further randomizes the bit value distribution of the obfuscated CRC. When intercepting a ciphertext, the attackers cannot determine the correct positions and thus cannot modify the original message at will.

B. Replay Attack

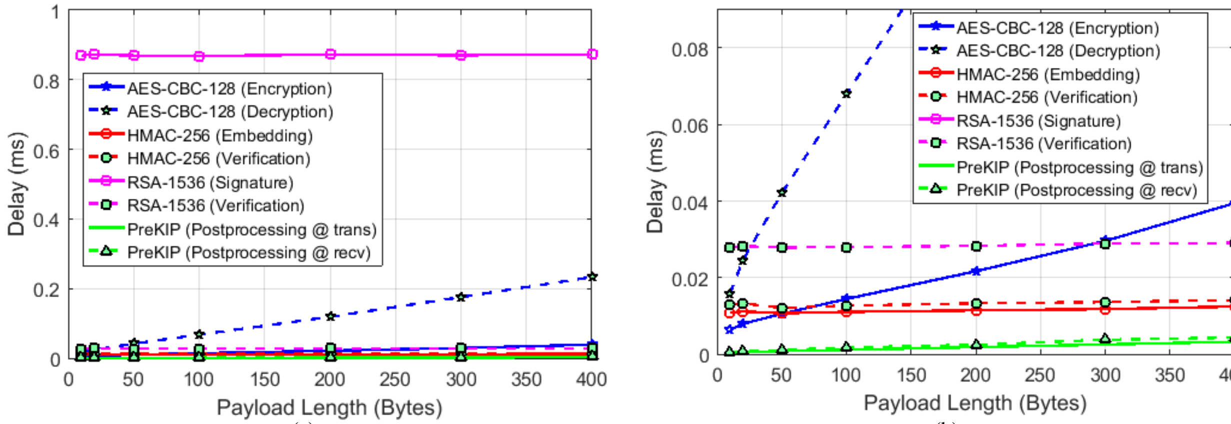
The adversary can launch a *replay attack* by overhearing a legitimate message and replaying it at some later time. In PreKIP, as the hashing is salted by incorporating the state number, sequence number, and a counter, a replayed message can easily be identified and discarded at the control center. The adversary may only succeed if he replays a message within a short time window of its origin, however, such an attack will not have a detrimental impact on knowing the current state of the grid and it cannot perturb the state as it would be easily detected and discarded.

V. Performance Evaluation

One of the key issue in the performance of an integrity protection algorithm is the resource-constrained microprocessors used by PMUs. Although the power of these processors is increasing, it will always be limited because of the small form factors. In this section we show that PreKIP is substantially faster as compared to other well-known cryptographic algorithms and can even meet the timing requirements of IEEE C37.118 when implemented on an inexpensive microprocessor such as LPC2148 [17].

Comparison between PreKIP and other schemes: We first compare between PreKIP and other well-known security and integrity techniques are summarized them in Fig 7. The results are obtained from a Intel Core i5-6500T @ 2.50GHz processor with 16 GB RAM, with compiler optimizations set to minimize execution time. All the simulations are averaged over 100 runs. In PreKIP we define the post-processing stage as the stage after the event (or packet) arrival, which includes CRC computation, obfuscation and encryption. The post-processing stage on the subscriber side is symmetric.

In Fig 7 we have compared the post-processing stage of our proposed PreKIP scheme along with (a) RSA-1536 (1536 bit RSA for public key crypto-system), (b) AES-CBC-128 (128 bit AES with block chaining) and (c) HMAC-256 (256 bit hash based MAC). Fig 7(b) is merely a zoomed version of Fig 7(a) for better visibility. We first compare RSA with other schemes. Not surprisingly, HMAC based signature generation is $\sim$80 times faster than RSA, whereas at the time of verification HMAC is $\sim$2 times faster than RSA. The results is also similar to some studies in [18], [19]. As compared to AES encryption, HMAC signature generation is $\sim$3x faster with a payload size of 400 Bytes. However, AES decryption performs



(a) (b)

Fig. 7. (a) Comparison of PreKIP along with AES, HMAC and RSA, along with its (b) enlarged view.

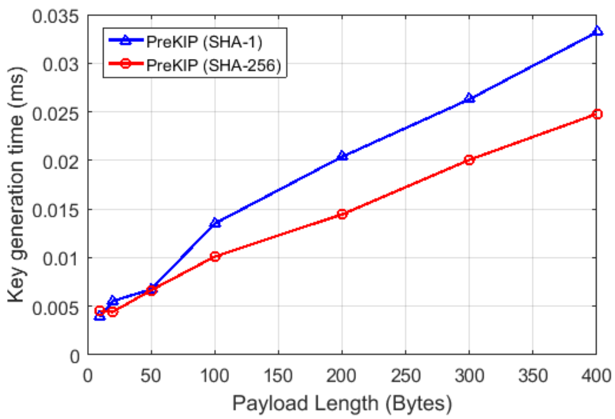


Fig. 8. The key-stream generation time with SHA-1 and SHA-256.

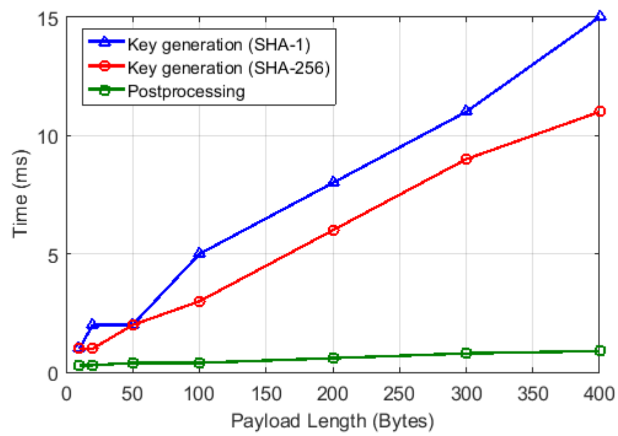


Fig. 9. Processing time of PreKIP for LPC2148.

poorly which results in ∼16x slower decryption than HMAC verification.

While comparing between PreKIP and HMAC-256, we observe that PreKIP is roughly 21x faster with a payload length of 10 Bytes, and ∼4x faster with 400 Bytes payload, than its nearest competitor HMAC-256, and even more as compared to others. In fact, PreKIP only takes 3–4μs in the post-processing stage with is much lesser than the inter-packet transmission time of SV (i.e. 173–416 μs) and the minimum retransmission interval of GOOSE (i.e. 0.5–5 ms). Notice that in Fig. 7 HMAC embedding and verification takes upto 12μs and 14μs respectively, which also fulfils the timing requirement of both SV and GOOSE. However, PreKIP is several times faster, and thus can be implemented even older/cheaper processors than HMAC. At the same time PreKIP also provides confidentiality of the information in addition to integrity. This experiment clearly shows the lightweight nature of PreKIP that makes it suitable for such low-latency cyrpto applications.

Key Generation Latency of PreKIP: Fig. 8 shows the key generation latency of PreKIP. We have compared SHA-1 and SHA-256 for generating the digests. From Fig. 8 we can observe that the key generation using SHA-256 is faster as compared to SHA-1. This is because of the fact that for a message with length L, and a digest length of length L_d, the hash function is called $\lceil \frac{L}{L_d} \rceil$ times. Even if SHA-1 is faster than SHA-256, the $\lceil \frac{L}{L_d} \rceil$ for SHA-1 is larger due to its smaller digest size (20 bytes as compared to 32 bytes in case of SHA-256). Hence, SHA-256 performs much faster than SHA-1 in key generation phase, especially for larger message length. Also notice that the key generation of PreKIP is actually slower than HMAC embedding. This is because of the fact that in PreKIP the hash function is called $\lceil \frac{L}{L_d} \rceil$ times, as opposed to just once in HMAC. However, this does not matter for smart grid message forwarding so long as the key stream is generated in between the samples (or retransmissions).

Performance of PreKIP on LPC2148: Fig. 9 shows the performance of PreKIP on LPC2148 microprocessor. LPC2148 ARM7TDMI microcontroller has a limited memory with 40 kB of on-chip static RAM and 512 kB of on-chip flash memory. Because of this limitation, it is not possible to implement arbitrary algorithms on it. Therefore, we have shown our evaluations only for PreKIP. Fig. 9 also shows faster key generation with SHA-256 as compared to SHA-1. Even with the largest message size of 400 bytes, the key generation is only 11 ms with SHA-256. With a maximum sampling rate of 60 samples/second, the total inter-sampling time is 16.67 ms, which is adequate for both key generation using the LPC2148 microprocessor. Furthermore, the post-processing stage is extremely fast, and just takes less than a millisecond. This shows the feasibility of implementing PreKIP on a low-end microprocessor for real-life applications.

From Fig. 9 we can also observe that LPC2148 microprocessor will not be suitable for SV or GOOSE message integrity checking, where sampling rate or minimum retransmission interval are 173–416 μs and 0.5–5 ms respectively. However, such high rates are unnecessary in IEEE C37.118 for transmitting PMU data where rates are limited to 30 to 60 samples per second (or 1/2 sample/cycle). In such applications, PreKIP can provide a secured communication even with an inexpensive microprocessor like LPC2148.

VI. Related Work

Security features have mostly remained unaddressed in IEEE C37.118 [20]. Whereas CRC based integrity check does exist in IEEE C37.118, such checks do not provide enough security to the messages and thus can be modified by the intruders. References [21]–[23] provide some study on cyber security challenges along with several potential threats in IEC 61850-substation network. To alleviate the security issues, IEC 62351-6 [24] recommends RSA digital signature algorithm for signing and verifying the substation messages. However, the timing related performances are studied in [18], [19], [25]–[27], which raised concerns over the applicability of RSA-based signature scheme for GOOSE messages. References [18], [19] have further studied that HMAC based provides faster integrity check as compared to RSA-based scheme and can satisfy the timing requirements for the GOOSE messages with today's commodity processors. In [28] the authors have implemented a framework for RSA and MAC based digital signature schemes to compare them. Reference [29] has presented a sequence hopping algorithm for securing the GOOSE messages. However, this requires a separate sequence synchronization and monitoring server, thus needs a separate infrastructure to be installed. As opposed to these contributions we have proposed a novel lightweight solution for the confidentiality and integrity problem mechanism, where the key is generated within subsequent message transfer, and thus the post-proceesing stage is 4–21 times faster than the HMAC scheme. The scheme also provides extra confidentiality and is secure from well-known attacks.

VII. Conclusion

In this paper, a lightweight and secure integrity protection algorithm PreKIP has been proposed to maintain the integrity of smart grid's measurements taken from the transmission and distribution networks. PreKIP achieves low latency by generating a new key between samples, and then simply XOR it with the generated sample. We show that our mechanism can meet the 3 ms delay target of IEC 61850 protection messages and is secure against powerful adversarial attacks. The proposed scheme is 4–21 times faster than the HMAC scheme while signing/encryption, and even faster than other crypto algorithms. In future we will explore the use of PreKIP in other applications (such as vehicle-to-vehicle communications in intelligent transportation systems) which require low latency for key generation, signing and verification.

References

[1] J. R. Carroll *et al.*, "A comparison of phasor communications protocols," Pacific Northwest National Lab.(PNNL), Richland, WA (United States), Tech. Rep., 2019.

[2] Y. Weng *et al.*, "Robust data-driven state estimation for smart grid," *IEEE Transactions on Smart Grid*, pp. 1–12, 2016.

[3] IEC 61850-90-5, "Use of IEC 61850 to transmit synchrophasor information according to IEEE C37. 118," 2012.

[4] S. R. Firouzi *et al.*, "Design, Implementation and Validation of an IEC 61850-90-5 Gateway for IEEE C37. 118.2 Synchrophasor Data Transfer," Ph.D. dissertation, Escola Tècnica Superior d'Enginyeria, 2015.

[5] Stefan Nohe, Oliver Hartmann, Farel Becker and Cedric Harispuru, "Designing non-deterministic pac systems to meet deterministic requirements," http://rtpis.org/psc13/files/PSC2013_final_1358893882.pdf.

[6] Herbert Falk, "Iec 61850-90-5 - an overview," https://www.pacw.org/issue/december_2012_issue/iec_61850905_an_overview/iec_61850905_an_overview/complete_article/1.html.

[7] Y. Fan *et al.*, "A cross-layer defense mechanism against GPS spoofing attacks on PMUs in smart grids," *IEEE Transactions on Smart Grid*, vol. 6, no. 6, pp. 2659–2668, 2015.

[8] Fred Steinhauser, "Route to IEC 61850 (2016): Client/Server, GOOSE and Sampled Values," https://www.youtube.com/watch?v=DuBayMLGrDY.

[9] "IEC 61850 Sampled Values protocol," https://www.typhoon-hil.com/documentation/typhoon-hil-schematic-editor-library/References/iec_61850_sampled_values_protocol.html.

[10] "IEC 61869-9:2016," https://webstore.iec.ch/publication/24663, 2016.

[11] K. Martin *et al.*, "Exploring the IEEE standard C37. 118–2005 synchrophasors for power systems," *IEEE transactions on power delivery*, vol. 23, no. 4, pp. 1805–1811, 2008.

[12] M. Adamiak *et al.*, "Design of a priority-based load shed scheme and operation tests," *IEEE Transactions on Industry Applications*, vol. 50, no. 1, pp. 182–187, 2014.

[13] P. Kundu *et al.*, "Synchrophasor-assisted zone 3 operation," *IEEE Transactions on Power Delivery*, vol. 29, no. 2, pp. 660–667, 2014.

[14] L. K. *et al.*, "Active attacks on stream ciphers with cyclic redundancy checks (crcs)," http://www.cix.co.uk/~klockstone/crchack.htm, 2000.

[15] M. Stevens *et al.*, "The first collision for full sha-1," in *Annual International Cryptology Conference*, 2017, pp. 570–596.

[16] G. Leurent *et al.*, "From collisions to chosen-prefix collisions application to full sha-1," in *EUROCRYPT*, 2019, pp. 527–555.

[17] "Arm7 based (lpc2148) microcontroller pin configuration," https://www.elprocus.com/arm7-based-lpc2148-microcontroller-pin-configuration/.

[18] D. Ishchenko *et al.*, "Secure communication of intelligent electronic devices in digital substations," in *IEEE/PES Transmission and Distribution Conference and Exposition*, 2018, pp. 1–5.

[19] H. B. K. Ghada Elbez *et al.*, "Authentication of goose messages under timing constraints in iec 61850 substations," in *ICS-CSR*, 2019.

[20] I. Ali *et al.*, "Performance comparison of IEC 61850-90-5 and IEEE C37.118.2 based wide area PMU communication networks," *Journal of Modern Power Systems and Clean Energy*, vol. 4, pp. 487–495, 2016.

[21] J. Cai *et al.*, "Review of cyber-security challenges and measures in smart substation," in *ICSGCE*, 2016, pp. 65–69.

[22] A. Chattopadhyay *et al.*, "Toward threat of implementation attacks on substation security: Case study on fault detection and isolation," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 6, pp. 2442–2451, 2018.

[23] T. S. Ustun *et al.*, "A review of cybersecurity issues in smartgrid communication networks," in *ICPECA*, 2019, pp. 1–6.

[24] IEC TS 62351-6, "Power systems management and associated information exchange - data and communications security - part 6: Security for IEC 61850," 2007.

[25] F. Hohlbaum *et al.*, "Cyber security practical considerations for implementing iec 62351," in *PAC World Conference*, 2010.

[26] S. M. Farooq *et al.*, "Performance Evaluation and Analysis of IEC 62351-6 Probabilistic Signature Scheme for Securing GOOSE Messages," *IEEE Access*, vol. 7, pp. 32 343–32 351, 2019.

[27] S. S. Hussain *et al.*, "Analysis and implementation of message authentication code (mac) algorithms for goose message security," *IEEE Access*, vol. 7, pp. 80 980–80 984, 2019.

[28] S. M. Farooq *et al.*, "S-GoSV: Framework for Generating Secure IEC 61850 GOOSE and Sample Value Messages," *Energies*, vol. 12, no. 13, pp. 1–13, 2019.

[29] Osama Mohammed, "Security aware microgrids:securing goose messages against cyberattacks," in *American Public Power Association and Florida Municipal Power Agency Southeast Regional Municipal Utility Cybersecurity Summit*, 2019.