# Saving Nine Without Stitching in Time: Integrity Check After-the-fact

Racin Gudmestad*, Siv Hilde Houmb†, and Martin Gilje Jaatun*‡
*University of Stavanger, Norway
†Statnett SF, Oslo, Norway
‡SINTEF Digital, Trondheim, Norway

***Abstract*—Electrical substations transform voltage from high to low, or low to high for distribution and transmission, respectively, and are a critical part of our electricity infrastructure. The state of a substation is continuously measured for monitoring, controlling and protection purposes, using synchrophasor measurements. The IEC 61850 standard defines communication protocols for electrical substations, including synchrophasor measurement transmission. However, IEC 61850 does not properly address cyber security, leaving this critical infrastructure highly vulnerable to cyber attacks. This paper describes the development and testing of a novel mechanism for delayed integrity check for synchrophasor measurements. The results show that the solution manages to detect when integrity of the synchrophasor transmission is compromised, without adding any overhead or delay to the time-critical synchrophasor transmission itself.**

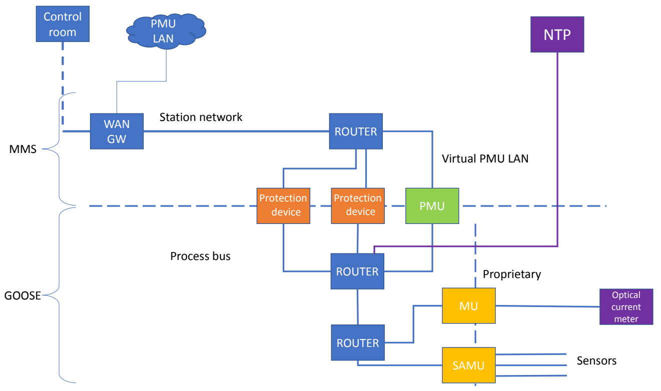


Fig. 1. Example digital substation with PMU

## I. Introduction

Substations all over the world are going digital, but experiences from Ukraine [1] have taught us that this is fraught with danger if cyber security does not receive due attention in the process. Since the communication protocols for industrial control systems (ICS) used in substations were not originally designed with cyber security in mind, the attackers did not have to find security vulnerabilities in the protocols to design the modular malware known as "Industroyer" [2] or "Crashoverride" [3]. They used their knowledge of the ICS environment, implemented the malware to interact with switches and circuit breakers at the substations using the insecure communication protocols, and launched their cyber attack causing the 2016 Ukrainian blackout. The blackout affected fewer subscribers than the previous cyber attack on the Ukrainian power grid in 2015, and only lasted a bit more than an hour, but recent analysis [4] shows that it had the potential of causing a much bigger and more serious disruption with possible physical destruction.

Protection devices within the digital substation have improved and benefited greatly from digitization. It is now possible to digitize current and voltage signals, and to send this data to protection devices that can react to it within just a few milliseconds. Perhaps the most important device responsible for generating and sending data to protection devices, is the Phasor Measurement Unit (PMU). The data it generates is (among other things) used to make sure that the grid's supply and demand are perfectly matched. Imbalances between the two can cause damage to the substation equipment and in worst case power outages [5]. It is absolutely essential for the control and protection mechanisms that maintain the balance between supply and demand, that the data from the PMU is correct and delivered with as little delay as possible.

From a cyber-security perspective, ensuring extremely fast data-delivery and integrity at the same time is quite a challenge. The cryptographic functions typically used to provide information security also increase the data and processing overhead, thereby increasing the delivery time of each message.

## II. Related work

### A. *Cyber security in synchrophasor transmission*

With the increased use of network technology, Intelligent Electronic Devices (IEDs), and software for monitoring and controlling substations, the overall attack surface has increased significantly [6]. With cyber attacks against power grids becoming more prevalent, it is crucial that critical infrastructure such as synchrophasor systems are properly secured.

Synchrophasor systems enhance the real-time monitoring and analysis of the power grid, and change the view of the power system from an estimated state, to a real-time directly measured state [7]. Today, Synchrophasor systems are not critical for all power system operations. This is either because the systems are in the experimental stages, or they only complement existing systems. However, there is a upward trend in the use of synchrophasor systems in critical functions such as in Wide-Area Monitoring Systems (WAMS) and Wide-Area Protection and Control Systems (WAPCS) [7]. There is no doubt that synchrophasor systems will become a critical part of the future power grid.

Measurements from PMUs, Phasor Data Concentrators (PDCs) and super PDCs are responsible for generating over 50% of the data that the WAMS and WAPCS use [8]. The protocols used to send the synchrophasor data are IEEE C.37.118.2 or IEC 61850 GOOSE messages. IEEE C.37.118.2 does not include any security mechanisms, whereas IEC has published IEC TR 61850-90-5 [9] (in the following referred to as TR 90-5 for brevity), which addresses security by suggesting security mechanisms for encryption and authentication of SV and GOOSE messages. However, these security features are applied to every single packet. Since the devices responsible for generating and receiving this traffic have limited hardware, the delay and overhead that this introduces is very high. Thus the solutions suggested in TR 90-5 are unsuitable for time critical real-time data transmission, e.g. synchrophasor transmissions.

## B. Possible cyber attacks against synchrophasor systems

When considering which types of attacks are plausible against synchrophasor systems, we have to consider both physical and remote access attacks. The entry points that can be exploited are the routers into the substation, the physical IEDs/PMUs/PDCs and the SCADA/HMI that control and communicate with the synchrophasor system. If the attackers managed to bypass the security mechanisms at these entry points, they can attack at any level, i.e., component-wise, protocol-wise or topology wise [10]. The type of attacks that threaten PMU networks are: Man-In-The-Middle attacks (MITM), Denial-Of-Service attacks (DoS), Packet analysis, data spoofing attacks and injection attacks.

*1) Packet analysis:* Packet analysis uses tools such as Wireshark to view packets sent on a network. This type of reconnaissance "attack" is not harmful in itself, but the information gathered could be used to discover vulnerabilities in the network, which could be used in a more severe attacks in the future (e.g. Denial of Service (DoS) attack). Since the communication in the PMU network is unencrypted, the content of the GOOSE messages sent is susceptible to packet analysis (also called sniffing). Attackers could use packet sniffers to spy on the network traffic and gather information such as source/destination IP address, protocol type, open ports, name and location of components, and data payload.

The best way to mitigate this type of attack is through encrypting the data, but this is not a valid option in the context of time-critical synchrophasor systems. The delayed integrity check does not prevent this type of attack.

*2) Denial of Service (DoS):* DoS are attacks that compromises availability by denying, or reducing the quality of service to a resource. DoS attacks are one of the most common threats towards synchrophasor systems [7]. An adversary who manages to gain access to an internal PMU network could easily perform a DoS attack simply by flooding the network with bogus traffic, overwhelming the switches/routers so that the legitimate traffic from the PMU is delayed or dropped. DoS attacks can also prevent critical control signals to come through.

A secure network infrastructure is the best defence against DoS attacks. This means multi-level protection, with intrusion detection system in combination with VPN, firewalls, anti-spam, content filtering, and load balancing. The delayed integrity check itself does nothing to mitigate this type of attack.

*3) Man-In-The-Middle Attacks (MITM):* MITM is when an attacker impersonates two communicating devices and makes them think that they are communicating directly with each other. If the attacker manages to get access to the communication infrastructure, remotely or locally, the attacker can in theory disguise themselves as the PMU or PDC (spoofing), and alter the messages sent between the two. This compromises the integrity of the messages. The attacker can also choose to drop the messages (DoS attack), thus also compromising availability.

Since the PMU multicast GOOSE messages using a publisher/subscriber mechanism, disguising a subscribing PDC with the intention of intercepting and altering/dropping GOOSE messages won't work. However, this does not mean that MITM attacks are not a threat in the context of PMUs. Command and configuration messages are unicast messages sent between two communicating devices, and are thus susceptible to a MITM attack. An attacker who successfully manages to act as a MITM between a PMU and its control application could have full control of the configuration and behavior of the PMU. By intentionally misconfiguring the PMU, both integrity and availability can be compromise.

This issue is not within the primary scope of the delayed integrity check, and would require another solution. TR 90-5 [9] suggests using a key distribution center (KDC) and authentication mechanisms between all communication devices. In the context of control and configuration messages, this is a valid option that should be implemented.

*4) Packet injection:* There are two types of attacks that could be performed through packet injection: command injection and sensor measurement injection. Command injection is when false control commands are sent to the devices within the synchrophasor system. Sensor measurement injections is when false measurements data are injected, tricking the control algorithms to make the wrong decisions. The latter is the one relevant in the context of synchrophasor systems.

There is no authentication mechanism in either IEC 61850[11] (pre TR 90-5) or IEEE C37.188 [12], which means that the PDC or application blindly trusts the messages it receives to be from a genuine PMU. A packet injection attack on the synchrophasor communication is therefor quite easy to perform, as the attacker is not required to obtain or crack any credentials or keys. If an attacker manages to gain access to the communication network, he can easily inject packets. The attacker would however have to spoof the address of the PMU it wants to impersonate, since the PDC/application only listen for GOOSE messages with a certain source address.

The delayed integrity check does not provide a mechanism for authenticating the communication between a PMU and a PDC/application, so injecting packets is still as easy as before. What it does, however, is to detect when this type of attack occurs in an timely manner. The integrity check runs in parallel to the synchrophasor communication, and the devices used in the integrity check has an authentication mechanism between themselves and the PMU. This makes them able to detect when malicious traffic **not** generated by the PMU have been injected into the communication stream. Thereby detecting the attack and shutting down the PMU before any harm can be done, thus indirectly stopping the attack. To prevent this type of attack completely, one would have to implement an authentication mechanism between the PMU and all the devices it communicates with, as well as adding HMACs to every single GOOSE message. This is the suggested solution in IEC TR 90-5 [9], but as mentioned in the introduction to this chapter, the delay and overhead this approach it introduces is simply to high for real-time synchrophasor communication.

## *C. Existing Security Mechanisms in IEC 61850*

TR 90-5 addresses several aspects of security, with the following assumptions:

- Authentication and integrity of information is needed
- Confidentiality is optional

TR 90-5 states that information authentication and integrity should be provided in an end-to-end method, through the use of information/message authentication codes. Furthermore it suggests that confidentiality should be optional. To achieve this TR 90-5 points to the use of symmetric/asymmetric cryptographic functions and the use of Key Distribution Centers (KDC) to distribute keys. However, TR 90-5 has a few inconsistencies and misconceptions in its discussion regarding security, and it seems like the report has not been written or reviewed by experts in the field of security [13]. Even though the introduction to chapter 8: Security Model, states that it “provides specifications for asymmetric key authentication/MAC creation” [9], no specifications with regards to asymmetric cryptography is found anywhere within the report. TR 90-5 mentions the use of symmetric keys to create and verify signatures. Symmetric keys are used for MACs (Message authentication codes), while digital signatures require asymmetric keys. The term KDC is also used incorrectly, when the report states that each IED is its own KDC. When each IED can distribute keys, they are not KDCs, they just make use of direct key negotiation.

TR 90-5 leaves out important details in their discussion about security, e.g. how to perform initial key distribution, and key distribution to logical devices (PMU/PDC). This leaves the implementation up to the vendors, giving them freedom to implement it however they see fit, which is not optimal when the goal of the IEC 61850 standard is to achieve interoperability.

*1) Key distribution:* To provide both asymmetric and symmetric cryptographic support to the synchrophasor communication, key management is needed. TR 90-5 suggests the use of Key distribution centers (KDCs) to provide symmetric key coordination between publishers and subscribers. The normal KDC implementation is to deploy the KDC as a standalone unit/node, but TR 90-5 argues that this raises concerns in regards to redundancy and issues related to providing uninterrupted delivery of information [9].

Therefore, TR 90-5 suggests that each IED shall be its own KDC. To allow continuous information exchange, the KDC needs a mechanism for informing subscribers of an impending key exchange, and a mechanism that informs subscribers that a key change has occurred. This is accomplished through the TimetoNextKey and TimeofCurrentKey session attributes.

When TimetoNextKey reaches 0, the publisher starts using a new key. The subscriber interacts with the KDC to obtain the next key, when it detects that TimetoNextKey is a positive value. It then waits until it receives a PDU with changed TimeofCurrentKey in its session header, before using the newly acquired key. TR 90-5 recommends that the symmetric key-pairs are changed at least every 48 hours, and the configuration should allow the definition of a minimum and maximum time (with 48 hours as default max, and 30 minutes as default min).

*2) Authenticated encryption:* AES-GCM (Advanced Encryption Standard in Galois Counter Mode) with the option between 128- and 256 bit symmetric keys are proposed. AES-GCM is considered to be highly secure and is widely adopted due to its performance. By using one of these functions for authenticated encryption, both integrity and confidentiality is ensured. The extra data and processing overhead added to each message, compared to just adding a MAC is negligible. Therefore, one could say that you get confidentiality for free. However, one could also argue that there are cases where confidentiality is not desired, e.g. when network analysis tools are needed to inspect packages.

Just like with TimetoNextKey and TimeofCurrentKey, The TR 90-5 session protocol has a security information attribute in the session header called SecurityAlgorithm, which specifies which of the two modes are being used (or if encryption is used at all).

*3) MAC:* TR 90-5 incorrectly uses the terms MAC and signature interchangeably. A MAC uses a symmetric key, while a signature uses an asymmetric key pair. Since the report only specifies key distribution and MAC algorithms for symmetric keys, it is assumed that they actually mean MAC when they use the term signature. The allowed hash MAC (HMAC) functions specified are HMAC-SHA265 and AES-GMAC. As long as authenticated encryption is not being used it is mandatory to use one of these two MAC functions, otherwise, the option “none” should be used. That way integrity and authenticity is always ensured for every message.

*4) Hash functions:* On page 86, TR 90-5 shows a table of “secure signature hash algorithms” (should be MAC). Listed are MD5 and SHA-1, which both are deprecated and should not be used. However, on the section about MAC algorithms on page 84, SHA-256 is listed as a supported option. MD5 and SHA-1 have been deprecated, but SHA-256 is considered to be secure today, and could thus be used instead.

Even though TR 90-5 contains some ambiguities and misconceptions, to some degree it manages to describe a way to achieve both integrity and authenticity of synchrophasor communication. However, since important details are left out, it leaves critical design and implementation decisions up to each specific vendor. In addition, just because this is the only solution that TR 90-5 provides, it does not necessarily mean that it is the best solution. As mentioned several times earlier,

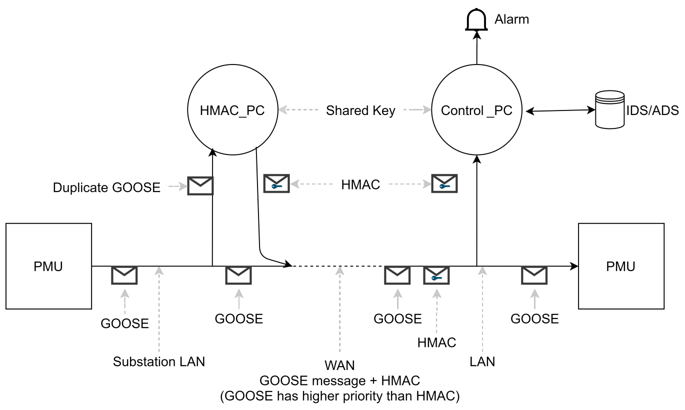


Fig. 2. Delayed integrity check concept

the increased processing and data overhead by adding MACs or encryption to each message is not desirable for time critical communication.

## III. Digital Substation Case

The conventional approach for assuring integrity is that the sender creates a hash-based message authentication code (HMAC) of the original message (OM), and appends this to the OM before sending it. The receiver hashes the OM with the same shared secret key and compares the HMAC it gets, with the HMAC that was appended to the OM. This is done for each message, and introduces too much data and processing overhead for a time critical real-time system.

This paper proposes an alternative solution, more specifically a delayed integrity check that processes the HMACs in a parallel process and raises an alarm in the case of mismatch. The goal is to develop and test whether it is possible to use a delayed integrity check for IEC 61850 GOOSE communication between Phasor Measurement Units (PMUs), without any (or with minimal) increase in the delivery-time of the PMU GOOSE messages.

## IV. Proposed Solution

Fig. 1 illustrates a simplified view of a digital substation. In reality, the process bus is duplicated, but this is not shown in the figure for simplification reasons. Our proposed solution is illustrated in Fig. 2. Briefly stated, regular PMU traffic is sent unmodified as before, but a special device (denoted HMAC_PC in the figure) makes a local copy of each GOOSE message and calculates an HMAC code which is sent separately (with lower priority) to the receiving system. On the recipient side, another unit (Control_PC) collects the HMAC value and the GOOSE packet, and verifies the correctness of the HMAC.

The Control_PC needs to buffer all GOOSE messages, and generate an alarm if either no HMAC message is received within a certain deadline, or if the associated HMAC is incorrect (indicating a tampered GOOSE message).

Here a PMU sends synchrophasor information over the process bus, using the GOOSE protocol. The figure shows three other devices that are all subscribing to the GOOSE messages from the PMU: HMAC-PC, Control-PC and PDC. The PMU and PDC acts as normal, where the PDC receives a stream of synchrophasor data from the PMU and merges it together with streams from other PMUs.

The HMAC-PC receives the same data stream as the PDC, since the PMU multicasts it out on the process-bus to all listening devices. The HMAC-PC uses a cryptographically secure hash algorithm (SHA-256) and a shared secret key to generate a HMAC. This HMAC is then sent from the HMAC-PC to the Control-PC (over a different VLAN). The Control-PC uses the same key and GOOSE message to create a HMAC, it then compares the HMAC it creates with the one sent from the HMAC-PC. By running the integrity check in parallel on two separate computers, there is no overhead added to the GOOSE messages.

This is in essence how the solution works; in the following we will explain in more detail.

### A. Solution design

Due to the limited time and resources at our disposal, this solution won't be implemented and tested at a real digital substation using PMUs, PDCs or a process bus. The goal of this work is to serve as a proof of concept for the delayed integrity check, which is why the solution is run and tested on regular computers over a LAN.

To test this solution, three different programs are run on three different computers, which are all connected on the same local network. One computer represents the PMU, one the HMAC-PC, and one the Control-PC. Since our purpose is to test this solution, there is no need for a fourth computer acting as a PDC.

The computer representing the PMU is running a simple python program. The program uses a Wireshark capture of real GOOSE messages as data source. The data is converted into a .txt file which the program reads and extracts real GOOSE messages from. Each individual GOOSE message in the txt file is added to an array list, which is iterated over. For each element (GOOSE message) in the list, a GOOSE message is extracted from it and sent over UDP to both the HMAC-PC and Control-PC.

### B. HMAC generation

A HMAC is a type of MAC that derives two keys out of the secret key (outer and inner) and uses two rounds of hashing. The first key is used together with the message to produce a inner hash. The second key is hashed together with the inner hash to produce the final HMAC. The length of the HMAC is therefore the same as that of the underlying hash function. The HMAC function is defined like this by RFC 2104 [14]:

$$HMAC(K,m) = H((K' \oplus opad)||H(K' \oplus ipad)||m))$$

- H is a cryptographic hash function (SHA-256)
- m is the message input to HMAC
- K is the secret key
- K' is a block-sized key derived from the secret key
- opad is the block-sized outer padding, consisting of 00110110 (36 in hexadecimal)

- ipad is the block-sized inner padding, consisting of 01011100 (5C in hexadecimal)
- || denotes concatenation
- $\oplus$ denotes bitwise exclusive or (XOR)

Python has a library called hashlib, which implements a common interface for many different secure hash and message digest algorithms. This includes HMAC generation with SHA-256 as the underlying hash function. HMAC-PC and Control-PC both use this library to generate HMACs.

For the integrity check to function properly, it is critical to figure out exactly how to perform the HMAC generation in terms of how often they should be generated and sent. If a HMAC is created for every single GOOSE message generated by the PMU, the traffic on the process bus would be doubled. Even though the time-critical GOOSE messages would have a higher priority than the HMACs and be sent over a different VLAN, the overall delay on the network would still increase due to the added load on the switches. An option could be to create HMACs based on randomly selected GOOSE messages, but this in turns opens a loophole which could potentially be exploited by attackers. An attacker could use the gap between two HMACs to send malicious GOOSE messages completely undetected.

Another solution could be to send a HMAC based on several GOOSE messages, by buffering messages and eventually performing the HMAC calculation on the whole buffer: HMAC = HMAC(K, Buffer). After n messages, the HMAC would be sent to the Control-PC for comparison. This reduces the load on the network, but introduces other issues. If a GOOSE message is lost, or delivered out of order, the HMACs would not match. And if n is set too high, an attacker could potentially send enough malicious GOOSE messages to trick the control mechanism to making a devastating ill-informed decision before the HMAC is sent and the attack is detected.

The solution provided in this paper is a combination of both "single" and "buffer" HMACs. Random GOOSE messages are picked out and used to create single-HMACs. To ensure that both HMAC-PC and Control-PC select the same random message each time, a shared secret key generated by a Diffie-Hellman key exchange is used as a seed into into a python function called "random.randrange()". Randrange() generates a random value between a given input range (1-250). This is a deterministic function, meaning that it will produce the same random value for both computers when they use the same seed. After a single-HMAC is generated using hmac.new() function, a new random value between 1-250 is selected. Then the seed is updated using a Blum Blum Shub (BBS) function with itself as input, so that the same value is not picked again.

The buffer HMACs are created with the same hmac.new() function, the difference is the input message. The buffer-HMAC uses a string consisting of the last 250 received GOOSE messages. Each new GOOSE message is appended to this string. HMAC-PC sends the buffer-HMAC for each 250th GOOSE message it received, and then resets the value of the HMAC-string. HMAC-PC adds an identifier to the HMAC, so that the Control-PC can differentiate between single and buffer HMACs. The Control-PC keeps updating its buffer-string until it receives a buffer-HMAC from the HMAC-PC, then it calculates its own buffer-HMAC with the string as input. After calculating and compering the two HMACs, the string is reset to nothing.

### C. *Alarm generation*

The whole point of this solution is to detect if the integrity between the communicating parties have been compromised. How to react if this is detected is not within the scope of this paper. As for this solution, if a mismatch between the HMACs is detected, the program simply outputs it into the terminal of Control-PC like this: "Single/Buffer HMAC detected for message with id X, mismatch rate = X%

## V. Experimental Results

### A. *Normal conditions*

When running a test of the solution as a whole, the scripts for the HMAC and Control computer are first run. They start by performing the initial key-exchange. An attacker that eavesdrops on the key-exchange can obtain all the information exchanged, but it is still considered to be infeasible to calculate the shared session key (seed). After the key exchange, the PMU script is run.

The number of single-HMACs sent between each buffer-HMAC varies. The HMAC and Control computer select "random" IDs between 1-250, and on average 3.18 single-HMACs are sent between each buffer-HMAC. The number 3.18 was found by simulation, using the same random functions and with n=10 000 000. A buffer-HMAC is generated for every 250th GOOSE message, but the gap between IDs is 1-466, and not 1-250. This is because the GOOSE messages in the Wireshark capture make up of about 53.8% of the data-set, with other traffic such as Sampled Value (SV) and Precision Time Protocol (PTP) making up the rest. The IDs are given sequentially to every packet, independent of the protocol used.

The sending rate of PMU.py is set to 0.015, to simulate the maximum sending rate of a real PMU(60Hz). Approximately 66 (1000ms/15ms) GOOSE messages are sent each second. The total time it takes to run a full simulation of the whole data-set is therefore equal to 86 minutes (66*60 / 345000 = 86.25 min). When running the program on a dedicated LAN, there is no package loss or out-of-order delivery of the GOOSE messages. Resulting in 0% mismatch rate for both single and buffer HMACs.

It is important to mention that even though the mismatch rate is 0% and there is no packet loss for this experiment, packet loss is a common phenomenon in modern networks and a real digital substation process bus would be no exception. Packet loss could potentially trigger a false-positive HMAC mismatch in a real substation.

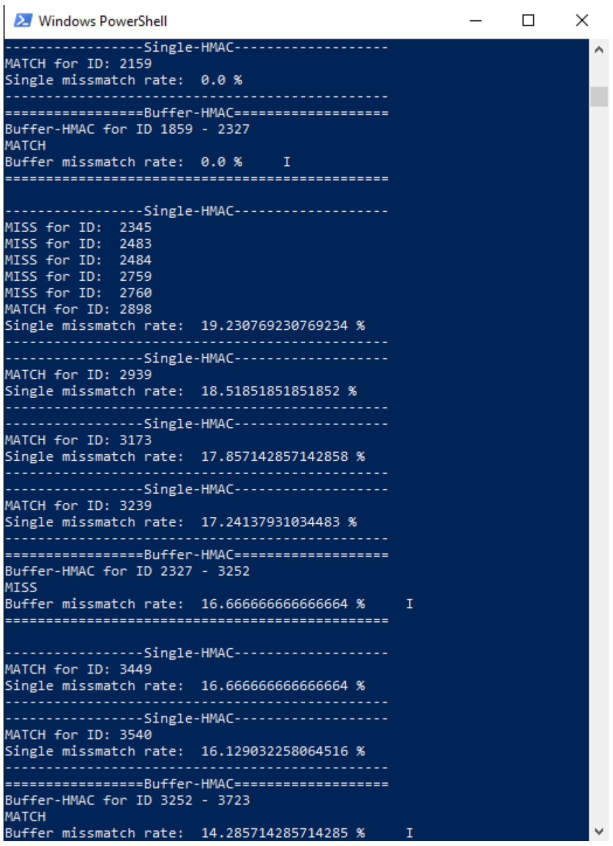
### B. *Simulated attack*

In this subsection two different variations of a packet injection attack will be simulated. In the first attack, a stream of 250 malicious GOOSE message will be injected sequentially. In the second attack, malicious GOOSE messages will be injected at random throughout the simulation.

To simulate the first attack, the following IF-statement have been added to PMU.py:

```
for i in range(0, length):
# For each GOOSE msg DO:
    if i > 1250 and i < 1500:
# Sends a packet ONLY to control pc
        pmu.sendto(GOOSE_Array[i],
         control_adr)
    else:
        pmu.sendto(GOOSE_Array[i], hmac_adr)
# Sends a packet to HMAC pc
        pmu.sendto(GOOSE_Array[i],
         control_adr)
        # Sends a packet to control pc
    time.sleep(0.015)
# max rate of PMU GOOSE messages
```

When running a simulation using this modified script, the GOOSE messages with ID 2345-2760 are only sent to the Control computer. As a result, the program outputs a series of single mismatches and a buffer mismatch for all the GOOSE messages within this range. Fig. 3 shows the output generated during this attack. After the injection attack is completed, the program goes back to running as normal and we can see the mismatch rate declining.



```
-----------------Single-HMAC------------------
MATCH for ID: 2159
Single missmatch rate:  0.0 %
----------------------------------------------
================Buffer-HMAC===================
Buffer-HMAC for ID 1859 - 2327
MATCH
Buffer missmatch rate:  0.0 %
==============================================

-----------------Single-HMAC------------------
MISS for ID:  2345
MISS for ID:  2483
MISS for ID:  2484
MISS for ID:  2759
MISS for ID:  2760
MATCH for ID: 2898
Single missmatch rate:  19.230769230769234 %
----------------------------------------------
-----------------Single-HMAC------------------
MATCH for ID: 2939
Single missmatch rate:  18.51851851851852 %
----------------------------------------------
-----------------Single-HMAC------------------
MATCH for ID: 3173
Single missmatch rate:  17.857142857142858 %
----------------------------------------------
-----------------Single-HMAC------------------
MATCH for ID: 3239
Single missmatch rate:  17.24137931034483 %
----------------------------------------------
================Buffer-HMAC===================
Buffer-HMAC for ID 2327 - 3252
MISS
Buffer missmatch rate:  16.666666666666664 %
==============================================

-----------------Single-HMAC------------------
MATCH for ID: 3449
Single missmatch rate:  16.666666666666664 %
----------------------------------------------
-----------------Single-HMAC------------------
MATCH for ID: 3540
Single missmatch rate:  16.129032258064516 %
----------------------------------------------
================Buffer-HMAC===================
Buffer-HMAC for ID 3252 - 3723
MATCH
Buffer missmatch rate:  14.285714285714285 %
```

Fig. 3. Attack 1 output: Injected stream

This illustrates that the integrity test works as intended. In a real digital substation, when such a dramatic increase in mismatches is observed, the protection mechanisms would be programmed to shut the PMU down to prevent the control applications that use the synchrophasor data from making a potentially devastating wrong decision. The decision to send out a signal that shuts a PMU down based on an increase in mismatched would be made by an ADS (Anomaly detection system).

In the second attack, GOOSE messages will be injected at random, instead of as a sequential stream of messages. PMU.py have been modified so that for each iteration (each sent GOOSE message) it selects a random value between 0-100. If the value is less or equal to 25, the GOOSE message is sent only to the control computer. This simulates an attack where an attacker tries to sneak malicious GOOSE messages into the process bus.

After running the simulation for a about 5000 iterations, the single mismatch rate stabilises around 25% as expected. The buffer mismatch rate remains at 100% throughout the simulation. This illustrates that the integrity check is capable of detecting this type of injection attack.

## C. Delay

Now that the integrity check have been proven to function as intended, it is time to look at how the solution performs in terms of delay. For the integrity check to be useful, it must be able to detect an attack fast enough. Fast enough in this context means that an attack should be detected before enough malicious message to make a control decision, have been received and applied by the control applications/mechanisms.

To figure out the time it takes to calculate a single HMAC, a simple test is run on PMU.py. This test creates a HMAC for each GOOSE message in the data-set, and calculates the average time it takes to create each one. The time it takes on a laptop with common CPU (Intel Core i7-7700 2.80 GHz CPU) is 0.00171 second. The same method has been used to calculate the time it takes for the Control-PC to compare two HMACs. The average time it takes for one HMAC comparison is 0.00095 seconds.

The next delay that needs to be taken into account is the time it takes to send the HMAC from the HMAC-PC to the Control-PC. Since this experiment have been carried out over Ethernet cables on a LAN with no other traffic, the delay in this experiment is not representable. Instead data from a 2017 study called "Analyzing Worst-Case Delay Performance of IEC 61850-9-2 Process Bus Networks Using Measurements and Network Calculus" [15] is used. The researchers found that the delay on their process bus was minimum 15.2 $\mu$-sec, average 16.5 $\mu$-sec and maximum 17.7 $\mu$-sec when sending SV messages from a merging-unit to a PMU. Since the message structure and protocol for both SV and GOOSE are similar, these numbers are assumed to be representable in this context.

It is also assumed that it takes approximately the same amount of time to send a HMAC from the HMAC-PC to the Control-PC, as it takes to send a GOOSE from the PMU to a PDC, since both are sent over the same fiber-optic process bus, and since the HMAC and Control PC is placed close to the PMU and PDC topology-vise. The average delay from when a GOOSE messages is received at a PDC/application, to the time a mismatch is detected is shown in figure 4

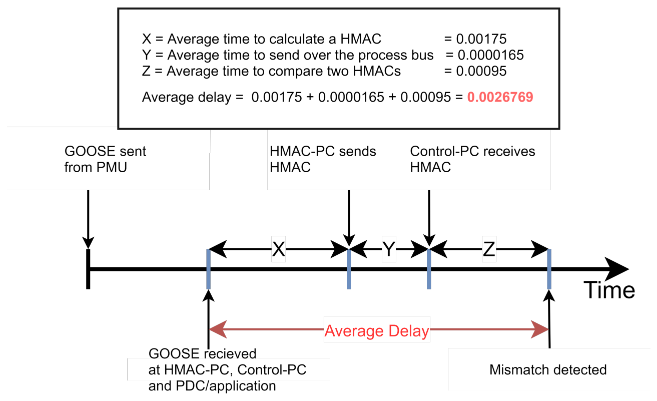

X = Average time to calculate a HMAC = 0.00175
Y = Average time to send over the process bus = 0.0000165
Z = Average time to compare two HMACs = 0.00095
Average delay = 0.00175 + 0.0000165 + 0.00095 = 0.0026769
GOOSE sent from PMU
HMAC-PC sends HMAC
Control-PC receives HMAC
X
Y
Z
Time
Average Delay
GOOSE recieved at HMAC-PC, Control-PC and PDC/application
Mismatch detected

Fig. 4. Average delay

With an average delay of 0.0026765 seconds and the maximum messaging rate of a PMU being 0.0166 seconds (1sec/60Hz), no new GOOSE message would have been generated and applied before a mismatch had been detected. Thus proving that the delay introduced by preforming the integrity check is not a limiting factor.

However, an attacker could (in worst case) get lucky, and start the injection of packets just after a single/buffer-HMAC match. The attack would last until the next single-HMAC is generated and the corresponding mismatch is detected. That way an attacker would not be able to send enough malicious messages to trick the control applications/mechanisms to make a wrong decision, before being detected.

## VI. Discussion

It might have been a viable option to generate HMACs for every single GOOSE message and use a separate VLAN to send the messages. It would be a simpler solution to develop, and it would be more secure since there is no gap between the single-HMACs that could be exploited. Since the integrity check is a parallel process running on a separate VLAN on dedicated computers, there is no processing delay or overhead added directly to the GOOSE traffic between PMU and PDC/application.

However, there is a delay added to the traffic indirectly. Even though the traffic is carried over a different VLAN, it still utilizes the same network infrastructure (e.g. fiber optic cables and switches). The assumption is that all this extra traffic generated by the HMACs would require a much higher resource usage from the switches, and thus increasing the overall latency on the network. However, if the switches would be able to handle the increased load without increasing the delay of the GOOSE traffic, this solution would be preferred over the single/buffer-HMAC solution. This is something that is hard to determine without testing both solutions thoroughly on real process bus implementation, but this is something we leave for future work. For now the assumption that the extra traffic would result in a higher latency and delay stands, and the random single-HMAC and buffer-HMAC method is perceived as the best option.

### A. *GOOSE re-transmission*

GOOSE messages are re-transmitted until a new event occurs, and the messages themselves contain a state number which tells the IEDs if the message is a re-transmission or a new message. When creating and testing the delayed integrity check, this re-transmission mechanism was not included in the PMU script, each GOOSE message was sent only once. This will be something that must be handled if the solution is to be deployed in a real substation. This could be done through keeping track of the ID of the last seen GOOSE message. For each message that has a state number which indicated that the message is a re-transmission, the ID in the re-transmitted message is compered to the last seen ID. If they match the re-transmitted message is discarded. The program on the HMAC and Control computer should also have a chronological list with the IDs of all the GOOSE messages that have been used to generate a HMAC. This list should be sent together with each buffer-HMAC to help determine the cause if there is a mismatch (e.g. packet loss, out-of-order delivery, injection).

### B. *Packet loss*

In section V-A, the mismatch rate was 0% for the buffer-HMAC when running a full simulation. On a real process bus it is highly unlikely that this would be the case, due to packet loss. Packet loss is a common phenomenon which would result in a false-positive buffer mismatch. However, this does not necessarily render the buffer-HMAC useless and untrustworthy. To deal with this issue, the process bus should be thoroughly tested and analysed before deploying the integrity check. This to figure out how high the average packet loss is on the channel, under all types of operating conditions. This information could be used to determine if a buffer mismatch is due to packet loss or not. If the mismatch rate is higher than what would be expected due to packet loss, it would give reason the believe that there are malicious GOOSE message being sent on the network.

Packet loss is usually caused by network congestion, problems with network hardware, software bugs and overloaded network devices. However, on a closed network such as a process bus, where everything is scaled and designed to handle more than the normal traffic load, network congestion and overloaded devices should in theory not be a concern. The packet loss is therefor assumed to be very low (closer to 0% than 1%). Buffer mismatches would happen due to packet loss now and then, but not so often that it becomes unreliable.

### C. *Anomaly based intrusion detection system*

To make the decision whether a mismatch is due to packet loss or malicious activity, an anomaly based intrusion detection system (ADS) could be used. An intrusion detection system (IDS) is a system that monitors network activity, classifying it as normal or malicious. An ADS is a type of IDS that looks for behavior/activity that deviates from the activity one would expect on a network, an anomaly. The system is thought what is normal behavior through a training phase, and uses this to build a profile for normal behavior. Once deployed, it compares the current traffic with this profile. If something deviates from it, like an increase in buffer mismatches or drastically increased GOOSE message, the ADS would be programmed to trigger an alarm that tells the PMU to shut down.

## VII. Conclusion

We have presented a solution for a delayed integrity check for IEC 61850 GOOSE communication. The solution offers a way to ensure integrity, without introducing any extra delay or overhead to the GOOSE traffic. We have shown that the proposed solution in TR 90-5 was not well suited for time-critical real-time data transmission such as GOOSE traffic, due to the increased overhead and processing delay. We have therefore presented an integrity check that runs in parallel to the GOOSE traffic, instead of as an integrated part of it. Thus, there was no need to make any changes to the existing GOOSE protocol, and there was no added overhead/processing delay to the synchrophasor transmission.

The delayed integrity check was implemented using python, and an experiment set up to serve as a proof of concept was carried out. The experimental setup consisted of three different computers (each running a different python script) connected to the same LAN, and a data set containing real GOOSE messages. The computers represent a PMU, HMAC-PC, and Control-PC within the substation. The result of this experiment showed that the program was able to detect when integrity was compromised, and did this fast enough to stop an attacker before any harm could be done to the substation.

### A. Further work

The goal of the program described in this paper was to serve as a proof of concept for the delayed integrity check. The program was tested on a LAN with a data-set of GOOSE messages. It is recommended to develop the integrity check further, so that it could be tested on a process bus in a real digital substation on real-time GOOSE traffic. Further investigation into the use of buffer/single-HMACs versus a HMAC for each GOOSE message should also be conducted.

The use of session keys with a maximum and minimum duration could also be implemented (like in TR 90-5). As of now, once a key is set, it stays in use until the program on both the HMAC and control computer is turned off and on again. Instead the key generated by the BBS function could be used as a session key. Once a predetermined timer runs out, the HMAC-PC switches to the next session key, and sends a message to the Control-PC informing it to do the same.

Lastly the solution should be integrated with an anomaly based intrusion detection system, for the purpose of generating alarms that quickly and correctly initiate preventive measures when an attack is detected. To aid the ADS in determining the cause of a mismatch, the following feature could be added to the integrity check: A list containing the IDs of all the HMAC-ed GOOSE messages could be sent from the HMAC-PC to the Control-PC together with the buffer-HMAC. If there is a buffer mismatch, the Control-PC would compare received list with a corresponding list of it's own. If the Control-PCs list is missing some IDs, it would indicate packet loss. If the Control-PCs list has more IDs than the HMAC-PCs, it would indicated packet injection. And if the lists have the same amount of IDs but in different order, it would indicate out of order delivery.

## Acknowledgment

The research reported in this paper has been supported by the FME Cineldi research project (257626/E20), funded by the Norwegian Research Council. This paper is based on the first author's MSc work at the University of Stavanger.

## References

[1] A. Cherepanov and R. Lipovsky. (2017) Industroyer: Biggest threat to industrial control systems since stuxnet. WeLiveSecurity by ESET. [Online]. Available: https://www.welivesecurity.com/2017/06/12/industroyer-biggest-threat-industrial-control-systems-since-stuxnet/

[2] A. Cherepanov, "WIN32/INDUSTROYER A new threat for industrial control systems," 2017. [Online]. Available: https://www.welivesecurity.com/wp-content/uploads/2017/06/Win32_Industroyer.pdf

[3] Dragos Inc. (2017) CRASHOVERRIDE – Analysis of the Threat to Electric Grid Operations. [Online]. Available: https://dragos.com/blog/crashoverride/CrashOverride-01.pdf

[4] J. Slowik, "CRASHOVERRIDE: Reassessing the 2016 Ukraine Electric Power Event as a Protection-Focused Attack," Dragos Inc., 2019. [Online]. Available: https://dragos.com/wp-content/uploads/CRASHOVERRIDE.pdf

[5] H. Haes Alhelou, M. E. Hamedani Golshan, T. Njenda, and P. Siano, "A survey on power system blackout and cascading events: Research motivations and challenges," *Energies*, vol. 12, 02 2019.

[6] D. U. Case, "Analysis of the cyber attack on the ukrainian power grid," *Electricity Information Sharing and Analysis Center (E-ISAC)*, vol. 388, 2016.

[7] C. Beasley, G. K. Venayagamoorthy, and R. Brooks, "Cyber security evaluation of synchrophasors in a power system," in *2014 Clemson University Power Systems Conference*, March 2014, pp. 1–5.

[8] M. D. Hadley, J. McBride, T. Edgar, L. O'Neil, and J. Johnson, "Securing wide area measurement systems," Tech. Rep., 2007. [Online]. Available: https://www.energy.gov/sites/prod/files/oeprod/DocumentsandMedia/8-Securing_WAMS.pdf

[9] IEC, "IEC TR 61850-90-5:2012 communication networks and systems for power utility automation - part 90-5: Use of IEC 61850 to transmit synchrophasor information according to IEEE c37.118, IEC std. 61 850-90-5, 2012-05." pp. 1–148.

[10] Y. Lu, M. Jafari, P. Skare, and K. Rohde, "An integrated security system of protecting smart grid against cyber attacks," 02 2010, pp. 1 – 7.

[11] IEC, "Communication networks and systems in substations - part 7-2: Basic communication structure for substation and feeder equipment - abstract communication service interface (acsi), iec std. 61 850-7-2 (first edition), 2003-05," pp. 1–148.

[12] "IEEE standard for synchrophasor measurements for power systems," *IEEE Std C37.118.1-2011 (Revision of IEEE Std C37.118-2005)*, pp. 1–61, Dec 2011.

[13] M. G. Jaatun and M. E. G. Moe, "PMU/PDC security considerations using IEC TR 61850-90-5," SINTEF memo, 2019.

[14] D. H. Krawczyk, M. Bellare, and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication," RFC 2104, Feb. 1997. [Online]. Available: https://rfc-editor.org/rfc/rfc2104.txt

[15] H. Yang, L. Cheng, and X. Ma, "Analyzing worst-case delay performance of iec 61850-9-2 process bus networks using measurements and network calculus," 05 2017, pp. 12–22.