Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Delayed Integrity Check for IEC 61850 Communication

Master's Thesis in Computer Science
by

Racin Gudmestad

Internal Supervisor

Martin Gilje Jaatun

External Supervisor

Siv Hilde Houmb

Reviewers

June 12, 2020

"Assumption is the mother of all fuckups"

Marcus Penn - Under Siege

Abstract

Electrical substations transform voltage from high to low, or low to high for transmission and distribution, and are a critical part of our infrastructure. The state of a substation is continuously measured for monitoring, controlling and protection purposes. The state is measured in real-time using synchrophasor measurements. The IEC 61850 standard defines communication protocols for electrical substations, including a standard synchrophasor transmission. However, IEC 61850 does not properly address cyber security, leaving critical infrastructure highly vulnerable to cyber attacks.

In this thesis, a delayed integrity check for synchrophasor measurements have been developed and tested. The result shows 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.

Acknowledgements

I would like to extend my profound gratitude to my supervisor Martin Gilje Jaatun. The weekly meetings, discussions, and feedback have been greatly appreciated. I would also like to thank my external supervisor at Statnett, Siv Hilde Houmb for her valuable help and insight. The tour at Statnetts pilot substation was highly informative and appreciated.

Contents

Abbreviations

IEC	**I**nternational **E**lectrotechnical **C**ommission
SCADA	**S**upervisory **C**ontrol **A**nd **D**ata **A**cqusition
SMV	**S**ampled **M**easurement **V**alue
GOOSE	**G**eneric **O**bject **O**riented **S**ubstation **E**vent
MMS	**M**anufactoring **M**essage **S**pecification
IED	**I**ntelligent **E**lectronic **D**evices
MU	**M**erging **U**nit
PMU	**P**hasor **M**easurement **U**nit
PDC	**P**hasor **D**ata **C**onsentrator
TCP	**T**ransmission **C**ontrol **P**rotocol
SCSM	**S**pecific **C**ommunication **S**ervice **M**apping
ACSI	**A**bstract **C**ommunication **S**ervice **I**interface
CDC	**C**ommon **D**ata **C**lass
KDC	**K**ey **D**istribution **C**enter
DOS	**D**enial **O**f **S**ervice
MITM	**M**an **I**n **T**he **M**iddle
MAC	**M**essage **A**uthentication **C**ode
IDS	**I**ntrusion **D**etection **S**ystem

Chapter 1

Introduction

The term digitization is perhaps one of the first words that come to mind when we talk about the technological trends that are changing and shaping society today. Digitization is the process of converting information into a digital format, which in turn a computer can read and process. It's about using technology to renew, simplify, and improve existing solutions. To make them more cost-efficient, reliable, and easier to use.

The energy sector is in the process of adopting this "new" trend. New and smarter equipment is being introduced into the traditional electrical substations. With this new equipment comes many advantages, like self-maintenance, reduced size, reduced wiring, improved integrity, reliability and availability, and many-in-one devices that are plug-and-play. But with new equipment comes new challenges, such as interoperability and cyber-security. IEC 61850 [1] is an international standard that defines communication protocols for devices inside digital substations. One of the main goals of IEC 61850 is to tackle the interoperability challenges between substation equipment from different vendors.

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 many 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 [2]. 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 the 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.

Information security involves confidentiality, availability and integrity of data. In the context of real-time PMU data (or synchrophasor-data as it is referred to in IEC 61850), integrity is the most important security feature. Availability is second, and confidentiality bears least importance of the three (more details in chapter 3.2). But even though providing confidentiality and availability is of less importance, the data and processing overhead generated by just assuring integrity is considered to be too high. Some of the protection functions that rely on synchrophasor data cannot tolerate more than a few milliseconds delay, which is why there is a need for an integrity check that does not impact the delivery time of the PMU messages at all.

1.1 Objective

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 thesis 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 of this thesis 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. Figure 1.1 shows the overall idea of the solution approach. This figure will be explained in detail in chapter 3.

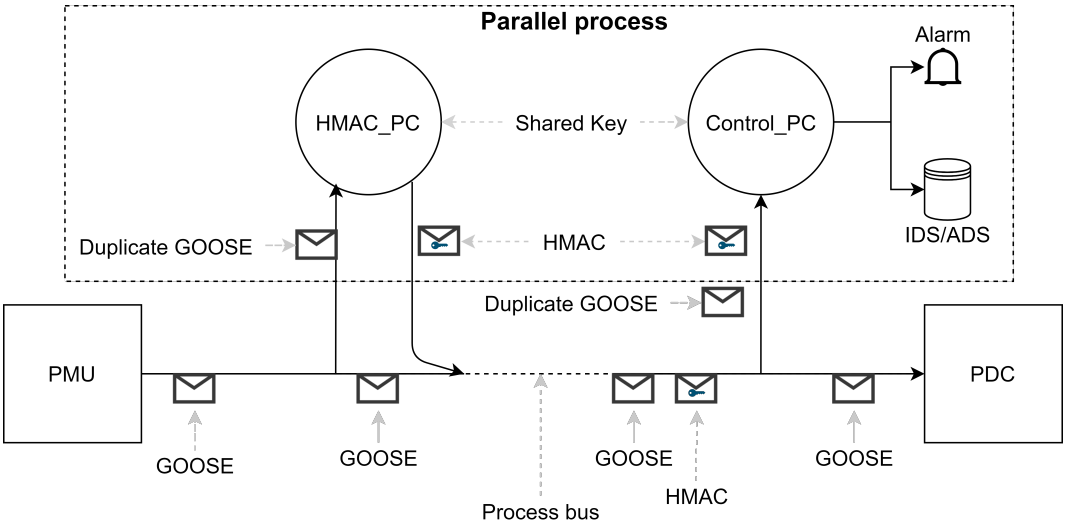

Figure 1.1: Solution model

1.2 Use-case

Statnett is an enterprise owned by the Norwegian Ministry of Petroleum and Energy. They own, operate, and build the Norwegian transmission grid. In 2017, Statnett (together with multiple partners) started a research and development project called "Digital Stasjon" [3]. The goal of the project is to build a digital substation pilot and to harvest knowledge and experience form how it works together with Statnett's existing substation equipment. The substation is located in Furuset, Oslo.

Through this project, Statnett will be able to gain experience on how the concept digital process bus works in practice, and about functionality, compatibility, and coexistence with existing installations and equipment. The delayed integrity check that this thesis proposes is thought to serve as a proof of concept, that if proven to function as intended will be taken further and adopted in Statnetts own digital substation.

1.3 Challenges

The main challenge is to develop a delayed integrity check that is compliant with the IEC 61850 standard, and with the existing PMU network deployed in Statnett's test substation. The secondary challenge is to develop the integrity check in such a way that it achieves its goal of assuring integrity, without introducing any communication delay between the PMU and PDC.

1.4 Thesis outline

Chapter 2 - Background
This chapter introduces the relevant theory in relation to digital substations, substation equipment and IEC 61850

Chapter 3 - Security
This chapter gives an introduction to the security objectives integrity, availability, and confidentiality in the context of synchrophasor systems. It also gives examples of possible attacks against such systems.

Chapter 4 - Solution Approach
This chapter presents a detailed analysis of the existing solutions and their restrictions. As well as how the solution that this thesis introduces works, and how it is implemented.

Chapter 5 - Experimental Evaluation
This chapter shows how an experiment designed to verify and test the solution was set up, as well as the result of this experiment.

Chapter 6 - Discussion
This chapter discusses packet loss, false-positives mismatches, anomaly-based intrusion detection systems (ADS), and GOOSE re-transmission

Chapter 7 - Conclusion and Future Work
This chapter provides a summarizing conclusion of the thesis, as well as suggestions for further work.

Chapter 2

Background

This chapter will provide the necessary background theory that is relevant to this thesis. The chapter gives an introduction to electrical substations, both conventional and digital, and the relevant equipment it houses. Furthermore, it gives an introduction to the IEC 61850 Standard.

2.1 Electrical Substations

An electrical substation is the part of a power system where voltage is transformed from high to low, or low to high for transmission, distribution, and switching. The power transformer, circuit breaker, bus-bar, insulator, and lightning arrester are the main components of an electrical substation [4]. In addition to all the physical equipment out in the switchyard, the substation has a control room/building from where the station is monitored and controlled. The physical components in the substation are connected to the control facility, which in turn is connected to the operation center via SCADA (Supervisory Control And Data Acquisition). Components like the transformers and circuit breakers are controlled from the operations center. From there circuit breakers can be opened and closed, transformers can be turned on/off and the turns ratio (whether the voltage is transformed from low-to-high or high-to-low) can be adjusted.

Values like voltage, current, and temperature are measured in each transformer, and the data is sent to the SCADA and control facility. In the control facility, there is also protection equipment, which is used to protect the components and cables in the substation if any error should occur in the power grid. For example, disconnecting components in the event of lighting or system overload [5].

In conventional substations, the connection between components, protection, and control equipment has been through extensive copper wiring. Each measurement has its own copper wire and the data is sent through analog electrical signals. As time has passed, new and smarter components and measuring equipment have been introduced into substations, which has reduced the need for this extensive psychical wiring.

2.2 Digital Substation

There are many different ways to define a Digital substation, one definition is that a fully digitized substation is a substation where as much as possible of the data related to the main processes is digitized immediately [6]. This includes but is not limited to measurements, control signals, and data exchange between units. The information exchange between units is done through a digital interface, and the functionality is first and foremost dependent on software.

Digital substations have major advantages in terms of design, engineering, installation, maintenance, cost, and operation compared to conventional ones. Off-the-shelf solutions can be offered and modifications are easy to accommodate. Cabling (and thus cost) is reduced, self-diagnostic units assure system integrity (and reduced maintenance cost), and in general, overall increased reliability and availability. Figure 2.1 shows the evolution of the substation from conventional to digital

2.2.1 Intelligent electronic devices

Another definition of a digital substation is an electrical substation where the operation is managed between distributed intelligent electronic devices (IEDs) interconnected by a communication network [8]. Intelligent Electronic Devices (IEDs) are perhaps the most revolutionary piece of equipment that has been introduced to the substation. An IED is, in essence, many devices put into one single device. Common IEDs include circuit breaker controls, protective relaying devices, capacitor bank switches, lead tap changer controls, re-close controllers, voltage regulators. IEDs receive data from sensors and power equipment and can issue control commands, such as tripping circuit breakers if they sense voltage, current, or frequency anomalies, or raise/lower voltage levels to maintain the desired level [9].

The IEDs can also communicate directly with the SCADA system. In addition to being many devices in one, IEDs have features like real-time synchronization of event monitoring, programmable logic controller functionality, self and external circuit monitoring, local

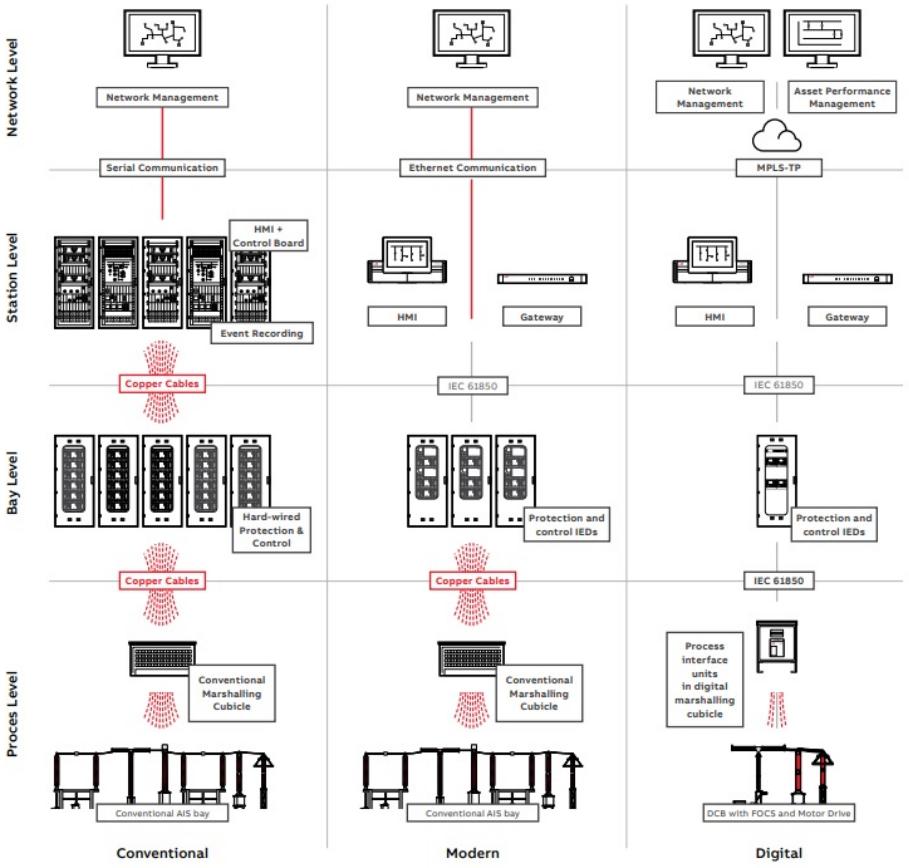

Figure 2.1: Substation evolution [7]

and substation data access, and a range of software tools for testing, event reporting, and fault analysis. IEDs can communicate with other devices over LAN/Ethernet.

2.2.2 Merging-Units

In digital substations, as much as possible of the data related to the main processes are instantly digitized, which means that all analog signals and measurements have to be digitized. In conventional substations, the measurements were sent as analog signals over copper wires. In digital substations, these measurements are digitized by the Merging-Units (MUs). MUs can digitize current and voltage signals and send them over the Ethernet network in the form of sampled values.

2.2.3 Phasor measurement units

A Phasor measurement unit (PMU) is a function within an IED (or a stand-alone unit) which is used to estimate the magnitude and phase angle of an electrical phasor quantity like voltage and current in the electricity grid. It uses a common time source

(e.g. GPS) for synchronization and sample values produced by MUs to generate the synchronized phasor (synchrophasor) measurement. A PMU can subscribe to a stream of sampled values one or many merging units within the substation. The synchrophasor measurements are used in real-time protection, monitoring, and control applications. The measurements are used to make sure that the grid's supply and demand are perfectly matched. Imbalances between the supply/demand can cause stress on the grid, which in turn can cause damage to the substation equipment and potentially cause power outages. Synchrophasor data is perhaps the most essential and critical data that is generated within a substation. Synchrophasor applications demand real-time transmission of messages with very low latency [10]. If for example, the demand in a power grid would drop drastically and quickly, the substation would have to adjust its supply almost instantaneously. This to avoid overload and physical damage to the grid and substation equipment. This is why the synchrophasor measurements are so time-critical and must have as little latency as possible. Some examples of the most important synchrophasor use cases/applications are listed below:

- Synchro-checking:
  - Checks the voltages upstream and downstream of a circuit breaker and allows closing when the differences in amplitude, frequency and phase are within authorized limits.
- Adaptive relaying:
  - The use of adjustable protective relay settings (e.g., current, voltage, feeders, and equipment) that can change in real time based on signals from local sensors or a central control system [11].
- Out-of-step (OOS) protection:
  - OOS occurs when two areas of a power system lose synchronism. The two areas must be separated quickly to avoid equipment damage or power outage.
- Situational awareness (SA):
  - Being aware of what is happening in the surroundings, and using this information to decide and act.

These are just some of the synchrophasor applications. Synchrophasor applications can be split into two main categories, online and offline. Online applications include real-time situational awareness for monitoring/WAMS (wide-area monitoring systems) purposes and are not as time-critical as offline applications which are mainly protection applications (E.g. Synchro-checking, Adaptive relaying and OOS).

2.2.4 Phasor Data Concentrator

A Phasor Data Concentrator (PDC) is a device that receives or subscribes to synchrophasor data from one or many PMUs and/or other PDCs [1]. The primary purpose of a PDC is to assemble all the input streams of synchrophasor data, into one single output stream. This is done to provide a more scalable distribution of synchrophasor measurements since PMUs are not designed to distribute data to a large number of clients due to their limited hardware. The stream is fed out to a higher level PDC or application [12]. The PDC data stream is used in the following functions: Situational awareness, state estimation, on-line security assessment, display, data processing, alarming, archiving, and wide area control [1]. Figure 2.2 shows a simplified model of how MUs, PMUs, and PDCs are connected

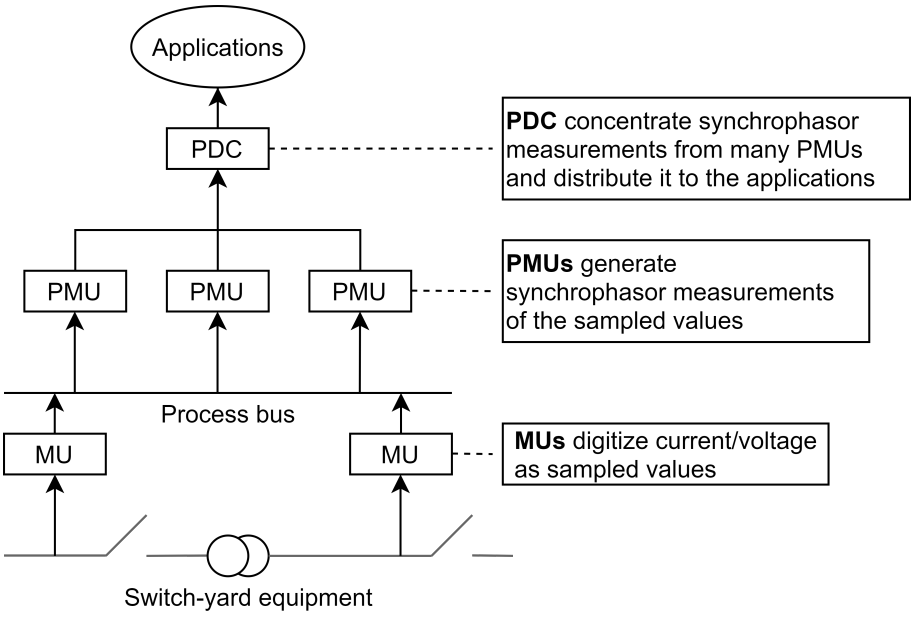

Figure 2.2: Simplified relation between MUs, PMUs and PDCs

2.3 Synchrophasor standards

2.3.1 IEEE C37.118

The first standard for synchrophasor data, IEEE 1344, was published back in 1995. IEEE did not address measurement accuracy, support for transmission hierarchy, software and hardware requirements, security mechanisms, transport protocol, or process for calculating synchrophasor measurements [10]. That standard was replaced in 2005 when "IEEE C37.118 - IEEE Standard for synchrophasors for Power Systems" was published.

IEEE C37.118 defined synchrophasors measurements used in power system applications, it provides a method to quantify measurements, tests to verify measurements, data communication protocols, messaging formats, and examples [13]. IEEE C37.118 only addressed accuracy for steady-state conditions. Over time, IEEE realized that they needed to address requirements for measurements under dynamic conditions [10]. To overcome this and to fix a few errors, an updated version was published in 2011. That standard was the result of a collaboration between IEEE and IEC, the goal of the collaboration was to determine how to progress the C37.118 standard in the context of IEC 61850. As a result, the C37.118 was split into two parts, IEEE C37.118.1-2011, and IEEE C37.118.2-2011.

IEEE C37.118.1-2011 standardized how to measure synchronized phasor data. The standard defined the terms synchrophasor, frequency, rate of change of frequency (ROCOF) measurement under all operating conditions [14]. IEEE C37.118.2-2011 specifies how to transfer synchrophasor data. The standard defines the messaging protocols, message types, data types, and data format used for real-time exchange of synchrophasor measurements between PMUs, PDCs, and applications. [12]

2.3.2 IEC TR 61850-90-5

The synchrophasor and how to transmit synchrophasor data over a network is defined in IEEE C37.118 and has been proven to work well. However, there is a need to have a communication mechanism that is compliant with the IEC 61850 standard [1]. IEC TR 61850-90-5 does exactly this, in addition to addressing cyber security requirements for encryption, key distribution, and hashed message authentication codes (HMAC). The IEEE C37.118 standard does not address security at all, and thus offers not a single security feature. IEC TR 61850-90-5 was published in 2012. The data structure, messaging/communication protocols, and security features that this TR 90-5 suggests will be analysed in detail in chapter 5.

2.4 IEC 61850 standard

An issue when building and designing something as complex as a digital substation is interoperability between substation devices from different vendors. IEC 61850 is an international standard for electrical substation automation and communication, developed to provide interoperability between IEDs from different vendors. IEC 61850 is a comprehensive standard, which was designed from the ground up to function over modern communications protocols. Interoperability in this context means that IEDs

from different vendors can share information and commands, and can operate on the same network. IEC 61850 enables integration of all protection, control, measurement, and monitoring functions, and additionally provides the means for high-speed substation protection applications, interlocking, and intertripping (controlled tripping of circuit breakers) [15].

The standard was initially intended for substation automation. However, after it was introduced, vendors and experts within the energy sector realized the benefits of a single international standard for utility systems. The latest releases have therefore been renamed from "Communication networks and systems in **substations**" to “Communication networks and systems for **power utility automation**”. In the following subsections, 4 parts of IEC 61850 will be discussed: Architecture, Communication, Data Model, and Engineering.

2.4.1 Architecture

IEC 61860 divides the substation into three levels (see figure 2.3). The first is the process level where we have switchgear such as circuit breakers and switches, and instrument devices like voltage and current transformers. The second level is the bay level. The bay level consists of IEDs that make decisions based on logic and information collected from the process level. IEC 61850 defines a process bus for communication between IEDs at the bay level and equipment at the process level. The current transformers (CTs) and voltage transformers (VTs) are connected to the process bus via merging units that act as an analog to digital converters. The third level is the station level. Here SCADA (Supervisory control and data acquisition) and HMI (human-machine interface) are used to control and monitor the substation. The station level and bay level are connected through the station bus. Both busses use high-speed fiber optic cables. Figure 2.3 shows a simplified model of the IEC 61850 substation architecture.

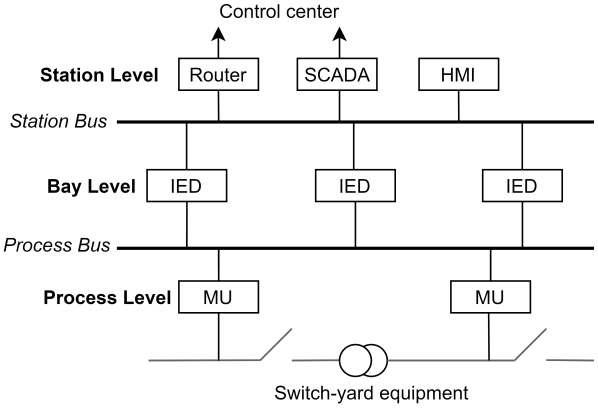

Figure 2.3: IEC 61580 Substation Architecture [16]

2.4.2 Communication

MMS

To enable fast and reliable exchange between substation devices, IEC 61850 has defined 3 protocols for substation communication. The first one and the most used is MMS, which stands for Manufacturing Messaging Specification. MMS is an international standard defined by OSI 9506 [17]. It is a client-server communication protocol where the client sends a request to the server, and the server sends back an answer. It allows for secure and private communication and is used between the IEDs and the SCADA/HMI systems to give status, make reporting, or send control commands.

GOOSE

The revolutionary breakthrough of IEC 61850 is the GOOSE message. GOOSE stands for "Generic Object-Oriented Substation Event" and is defined in IEC 61850 part 8-1 [18]. The protocol is used to send urgent messages on the process bus, such as synchrophasor data from PMUs/PDCs. The GOOSE message is directly linked to the data link layer of the OSI model and delivers messages straight to the application layer (See Figure 2.4). This makes it extremely quick and is why it is called a real-time protocol. GOOSE messages are delivered within 4 milliseconds and meet the high-speed performance requirements of automation and protection applications. The protocol is based on a mechanism called publisher/subscriber. The GOOSE messages are multicast over the LAN to all devices that are subscribing.

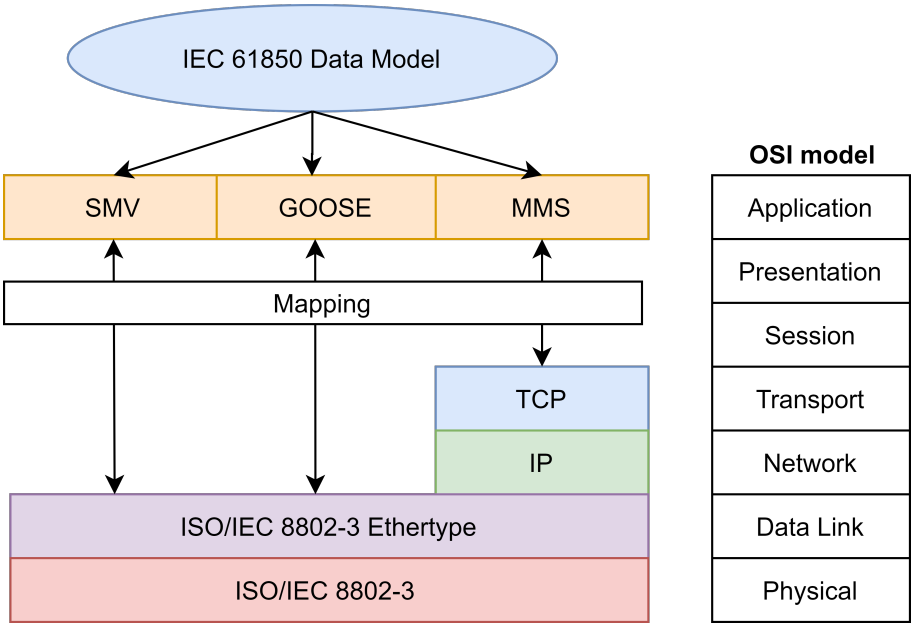

Figure 2.4: Mapping of IEC 61850 Data models into real protocols

Since all the layers between the Data Link layer and the application layer are skipped, there is no acknowledgment mechanism. GOOSE messages are urgent and cannot afford

to wait for acknowledgment for each message. Instead, the message is repeated within a certain time frame with increased frequency. GOOSE messages are event-driven, meaning that a new GOOSE message is only generated when something new has happened, e.g. a state changes. The same GOOSE message is re-transmitted, with increasing time between each re-transmission. The re-transmission of a message will stop when a new event triggering a new GOOSE has occurred. A state number within the GOOSE message identifies whether a GOOSE message is a new or re-transmitted message. This re-transmission mechanism also serves as a heartbeat function, telling all listening devices that the device is "alive" and transmitting. The re-transmission mechanism is illustrated in figure 2.5

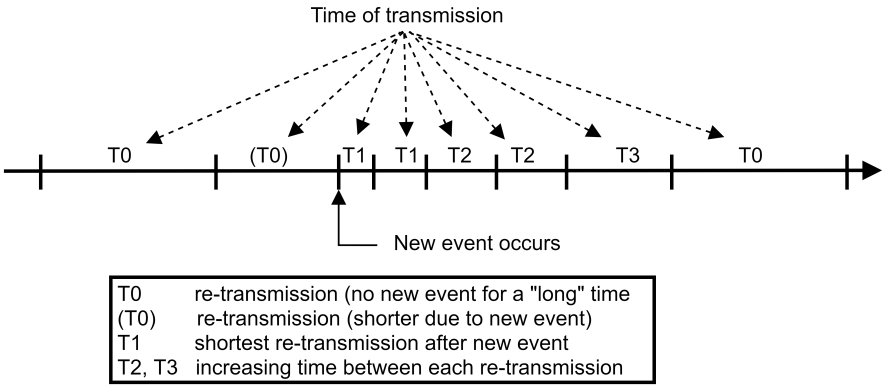

Figure 2.5: GOOSE re-transmission mechanism

Since the network layer is by-passed, GOOSE messages are unable to hop across networks/routers and thus stay within their local area network. An advantage of the link layer is that messages can be assigned VLAN flags/priority tags by the switches, and GOOSE messages are given a high priority due to their urgency. However, it is important to mention that there exists a protocol (R-GOOSE) that maps GOOSE messages into UDP-packets which can be sent across the internet. This is how synchrophasor data for online applications and PDCs on other networks is communicated.

As mentioned, every GOOSE message is multicast to all subscribing devices. The GOOSE message is "broadcast" with a multicast virtual group address as destination address [19]. This way the message is only forwarded to the subscribing devices. Using the standard broadcast address (FF-FF-FF-FF-FF-FF) would forwards the message to every device on the network. By multicasting to a virtual group, only IEDs interested in the GOOSE messages received them, thus preventing the overload of the network and IEDs that regular broadcast could cause.

The GOOSE protocol does not offer any security, GOOSE messages are sent in plain text without the option to use encryption or message authentication. This is addressed in

IEC TR 61850 90-5 [1] (in which R-GOOSE is defined). R-GOOSE has security features such as message authentication codes (MAC) and authenticated encryption.

The security features in R-GOOSE, and the structure of the GOOSE message will be explained in detail in chapter 5.

SMV

SMV (Sampled Measurement Value, often just referred to as SV) has many of the same characteristics as GOOSE. Both are directly linked to the data-link layer and use the Publisher/Subscriber mechanism to send messages. SMV is used for fast transmission of measurements generated by physical measuring equipment within a substation (such as a Merging Unit). These measurements are sent at regular intervals, not like GOOSE which is event-driven and sent upon a state change. A MU digitizes current and voltage signals and packs the measurements into SMV packets that are sent out on the process bus. These measurements are (among other things) what the PMUs/IEDs use to generate synchrophasor data which are sent as GOOSE messages (as seen in Figure 2.2). SMV is defined in IEC 61850 part 9-2 [20]

Redundancy Schemes

Due to the real-time nature of both GOOSE and SMV, standard redundancy configurations are not applicable, which is why IEC 61850 defines two schemes for redundancy: High-availability seamless redundancy (HSR) ring scheme and Parallel Redundancy Protocol (PRP) double start scheme. Both provide seamless failover of any network component and have zero recovery time with no information lost. Both are illustrated in figure 2.6.

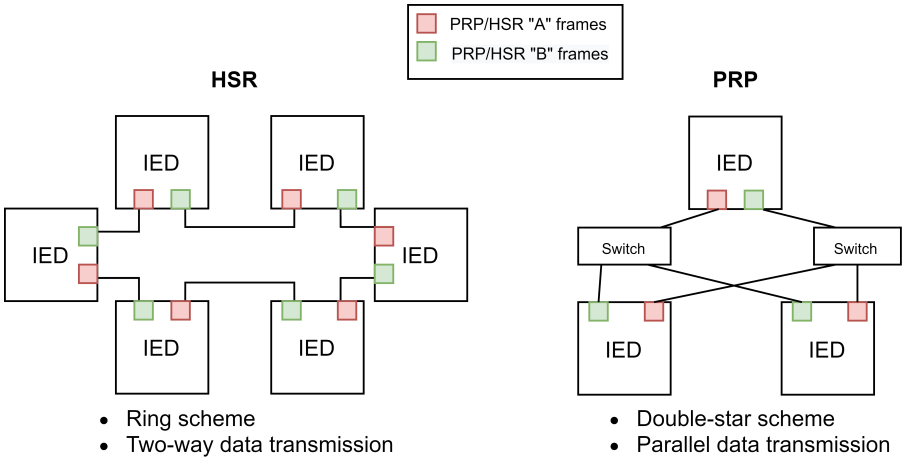

Figure 2.6: IEC 61850 redundancy schemes

2.4.3 Data model

IEC 61850 achieves interoperability by defining a standard data model for data exchange. All devices and functions have an object-oriented description. The model is described below:

- Physical device:
  - IEDs which have an IP addresses and are connected to the substation network. A physical device can consist of many logical devices
- Logical device:
  - "Entity that represents a **set** of typical substation functions" [21]. For example a PMU or Breaker controller
- Logical Node:
  - "Entity that represents a typical substation function" [21]. For example a Merging unit. Must follow a standard naming convention
- Data class:
  - Every logical node has one or more elements of data. Each data element belongs to a Common Data Class (CDC) which is defined in [21]. The classes use a standard naming convention to describe themselves, and each class contains several data attributes. The attributes belong to a function class, for example "POS" (position). IEC 61850 specifies if this attribute is mandatory, optional or conditional. Mandatory means that every vendor should implement this data attribute in their IEDs. The structure is shown in figure 2.7

Logical nodes and Data classes follow a standardised naming convention, while logical and physical devices do not. This abstract data and object model provides a standardized method for describing devices within a substation. This enables IEDs from different manufacturers to exchange data and information with the same format. ACSI (Abstract Communication Service Interface) is a virtual interface that the IEDs use to abstract communication services. E.g. connection, variable access, unsolicited data transfer, device control, and file transfer services [22]. IEC has defined Specific Communication Service Mapping (SCSM) which is a standardised procedure that provides a concrete mapping of ACSI services and objects onto a particular protocol stack/communication profile [23]. This service is illustrated in figure 2.4 (as mapping)

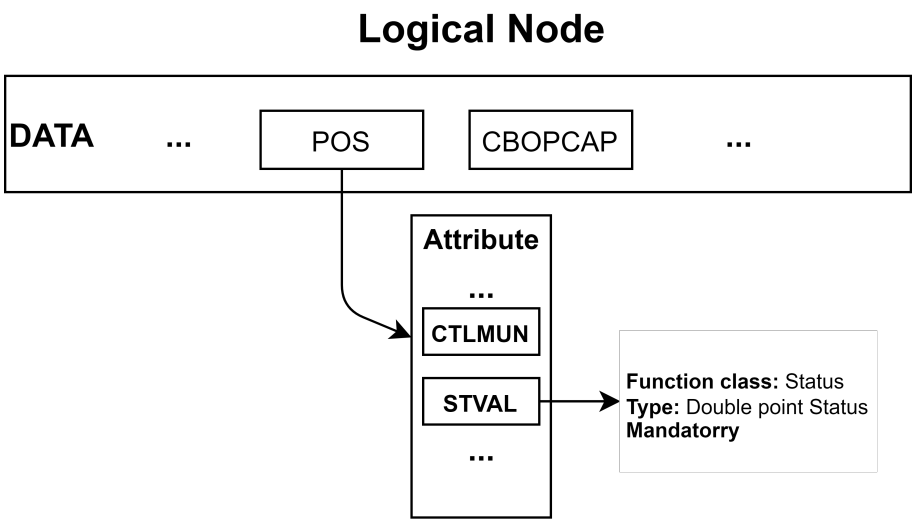

Figure 2.7: IEC 61850 Data structure

2.4.4 Engineering

The last part of the IEC 61850 standard is about engineering, which is an advanced part of the standard. From an engineering perspective, configuring a whole substation correctly requires a tremendous amount of work and is quite the challenge. To ease this task, IEC 61850 has defined something called Substation Configuration Language (SCL). SCL enables the configuration of a device and its role in the substation by using an XML (Extensible mark-up language) file. There exists tools/software that helps to create/write the different SCL files.

First, the single line diagram (representation of a power system, using symbols for each component) is translated into an SCL file, called substation Description file (SSD). All IEC 61850 compliant IEDs provide an IED capability description file (ICD). The SSD and ICD files are used in a system configurator software to create a new SCD file (Substation configuration description). During this phase, all interconnections between IEDs are also defined. The last step is to extract configuration IED descriptions files (CID) from the SCD file. The CID files are transferred to IEDs to automatically set their configuration.

2.4.5 IEC 61850 summary

IEC 61850 is a universal solution to engineering a fully digital solution for power utility automation. It is built in a way that can accommodate the future evolution of communication technology. It allows easy reconfiguration of the system without physical intervention on the ground, and it allows multi-vendor solutions. The standard is made up of several thousands of pages, this section has just scratched the surface of the parts relevant for this thesis.

Chapter 3

Security

3.1 Cyber security in synchrophasor transmission

With the increased use of network technology, IEDs, and software for monitoring and controlling substations, the overall attack surface has increased significantly. This is why there has been a dramatic increase in cyber attacks that target electrical utilities in the last five years. One example, and probably the most famous, is the cyberattack against the Ukrainian power grid in December 2015, when attackers shut down about 30 substations for 1-6 hours and left over 230 000 Ukrainians without electricity. Hackers managed to gain access to the Supervisory Control and Data Acquisition System (SCADA), by compromising corporate networks, and using spear-phishing emails to spread malware [24]. 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 [25]. 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 an 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) [25]. There is no doubt that synchrophasor systems will become a critical part of the future power grid.

Measurements from PMUs, PDCs, and super PDCs are responsible for generating over 50% of the data that the WAMS and WAPCS use [26]. 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, while IEC has published TR 90.5

[1], which addresses security. IEC TR 61850-90-5 has suggested 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 are very high. Thus the solutions suggested in TR 90-50 are unsuitable for time-critical real-time data transmission, e.g. synchrophasor transmissions. TR stands for "technical report", which means that IEC TR 61850-90-5 is not (as of right now) an official part of the IEC 61850 standard. It is to be viewed as suggestions, guidelines, or work in progress.

3.2 Security objectives

In the field of cyber security there are three security objectives: Confidentiality, Integrity, and Availability (CIA).

Integrity of data is the guarantee that data has not been altered, or that if the data has been altered, it is detected. The integrity of PMU data is the most important security objective since the data will be used to detect when crucial adjustments must be made. This applies to real-time data collected and transmitted by PMUs and PDC, and data stored in memory.

Availability ensures reliable, timely and uninterrupted access to information/data. It applies to communication systems, databases, applications, and networks. For a synchrophasor system, availability means that the measurements are continuously delivered on time, all the time. Availability is of high importance for synchrophasor systems, loss of availability would render the applications that rely on the data unable to perform their functions due to lack of information.

Confidentiality ensures that the data is not accessed or viewed by unauthorized parties. Even though PMU data might be interesting for some adversaries that want to know the current state of the power grid, it's not of as high importance as availability and integrity. PMU data is interesting for attackers doing reconnaissance and testing of their tools, but compromising the confidentiality won't have any direct impact, like compromising availability or integrity would.

Integrity is considered to be of higher importance than availability with regards to synchrophasor data. The reason for this is that if an attacker compromises the availability of PMU data, control, and protection mechanisms/applications that relies on this data won't be able to perform actions that control the grid. in worst-case, this could lead to the substation having to be controlled or shut down manually. If integrity is compromised however, an attacker could make the control mechanisms/applications make the wrong

decisions, thinking they are correct. This could cause power outages and shutdown of the substation, as well as physical damage to the substation equipment.

Worst-case example: imagine that an attacker injects synchrophasor measurements that show that there is an imbalance between supply and demand in the power grid. The control applications/mechanisms receive this false information and incorrectly make a decision to increase the supply. In reality, the supply is now much higher than the demand, but the control applications/mechanisms as well as the monitoring software/HMI show that the supply and demand are perfectly matched. This imbalance in turn causes an overload on the transformer, physically damaging it and causing it to shut down. By compromising integrity, the attacker has now successfully caused a power outage and damaged critical infrastructure, by just injecting false synchrophasor measurements into the system.

3.3 Possible cyber attacks against synchrophasor

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 [27]. 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.

3.3.1 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 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.

3.3.2 Denial of Service (DoS)

DoS is attacks that compromise availability by denying, or reducing the quality of service to a resource. DoS attacks are one of the most common threats towards synchrophasor systems [25]. 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.

Secure network infrastructure is the best defense against DoS attacks. This means multi-level protection, with an 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.

3.3.3 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 compromised.

This issue is not within the scope of the delayed integrity check and would require another solution. TR 90-5 [1] 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 (with a few improvements to the KDC, explained in chapter 4).

3.3.4 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 are 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[21] (pre TR 90-5) or IEEE C37.188 [14], which means that the PDC or application blindly thrusts the messages it receives to be from a genuine PMU. A packet injection attack on the synchrophasor communication is therefore 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 a 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 has 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 [1], but as mentioned in the introduction to this chapter, the delay and overhead this approach it introduces is simply too high for real-time synchrophasor communication.

Chapter 4

Solution Approach

As mentioned in Chapter 2, there are two standards that are being used today in synchrophasor communication: IEC 61850 and IEEE C37.118. Of the two, only IEC 61850 addresses security. This chapter will give a detailed analysis of the security features that IEC suggests in TR 61850-90-5, as well as a detailed description and analysis of the alternative solution that this thesis proposes with the delayed integrity check.

4.1 Existing Approach

A technical committee (TC 57) appointed by IEC has complied a technical report called "Part 90-5: Use of IEC 61850 to transmit synchrophasor information according to IEEE C 37.118" [1] (hence referred to as TR 90-5). A technical report is not an official standard. It is more of a work in progress that has yet to be included in the official standard. TR 90-5 is still in hearing, and it will be a while before it is fully adopted as a part of the IEC 61850 standard. This means that it is still susceptible to change.

That being said, TR 90-5 has still been implemented and is currently being tested in substations around the world. For example in [28], where TR 90-5 is implemented and compared to IEEE C37.118 in terms of performance and security. And in [29], where TR 90-5 is implemented and an IEEE C37.118.2 to IEC 61850-90-5 gateway is developed.

4.1.1 TR 90-5 security features

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 [30]. Even though the introduction to chapter 8: Security Model, states that it "provides specifications for asymmetric key authentication/MAC creation" [1], 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 the freedom to implement it however they see fit, which is not optimal when the goal of the IEC 61850 standard is to achieve interoperability.

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 [1].

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. Figure 4.1 illustrates the interaction between publisher and subscriber and the use of these two 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

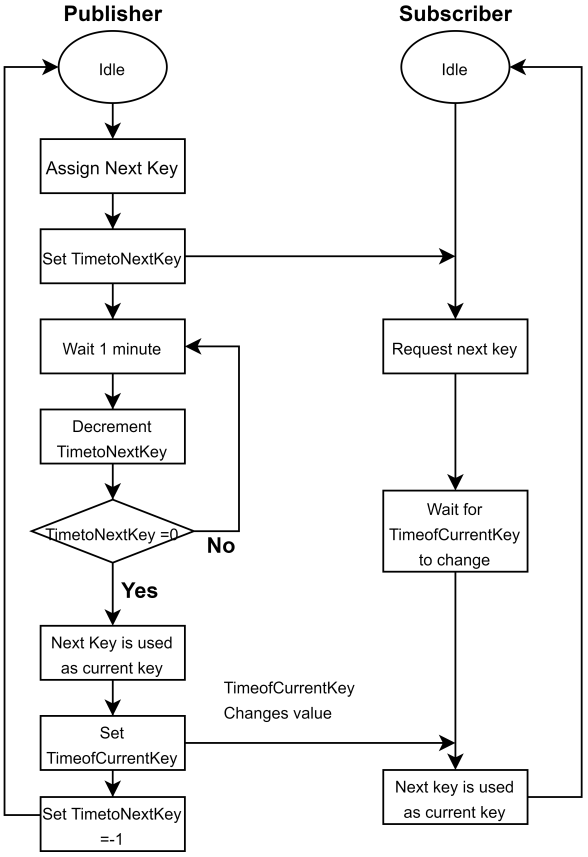

Figure 4.1: KDC key exchange scheme [1]

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).

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)

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 are always ensured for every message.

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, in the section about MAC algorithms on page 84, SHA-256 is listed as a supported option. This algorithm is not deprecated and 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, the increased processing and data overhead by adding MACs or encryption to each message is not desirable for time-critical communication.

4.1.2 Communication services

The IEC 61850 standard defines five communication mechanisms. Of these, the following three are used for time-critical information exchange: Sampled value (SV), Generic-Substation Event (GSS), and Generic Object-Oriented Substation Event (GOOSE). GOOSE is defined in IEC 61850-9-1 and is used for event-based data transfer. SV is defined in IEC 61850-9-2 and is used to transfer raw data/measurements generated by measuring equipment within the substation. As illustrated in figure 2.4 and explained in chapter 2.4.2, GOOSE and SV are directly linked to the data link layer and thus cannot be routed outside of the local network.

TR 90-5 introduces two options for routing SV and GOOSE packages over wide-area networks:

- Tunneling across a high speed communication network like SDH or Sonet [1]

- Over IP networks with UDP multicast

For the second option, TR 90-5 defines a new mapping of SV and GOOSE into the UDP protocol. This new mapping is what is known as routed-Sample value (R-SV) and routed-GOOSE (R-GOOSE). The application layer specifications of SV and GOOSE remain unchanged, however, TR 90-5 introduces a new protocol at the session layer to map GOOSE and SV into R-GOOSE and R-SV.

4.1.3 R-GOOSE and R-SV

TR 90-5 divides the OSI model into two profiles, Application (A-Profile) and Transport (T-Profile). R-GOOSE and R-SV are implemented at the A-profile. The A-profile specifies the communication profile that allows GOOSE and SV to be transported over an IP based network in a secure manner [1]. The T-profile covers the physical, data link, transportation, and IP layer of the protocol stack. Figure 4.2 show the encapsulation of the GOOSE/SV PDU and the A-profile and T-profile in relation to the OSI model. Figure 4.3 shows the A-profile in detail.

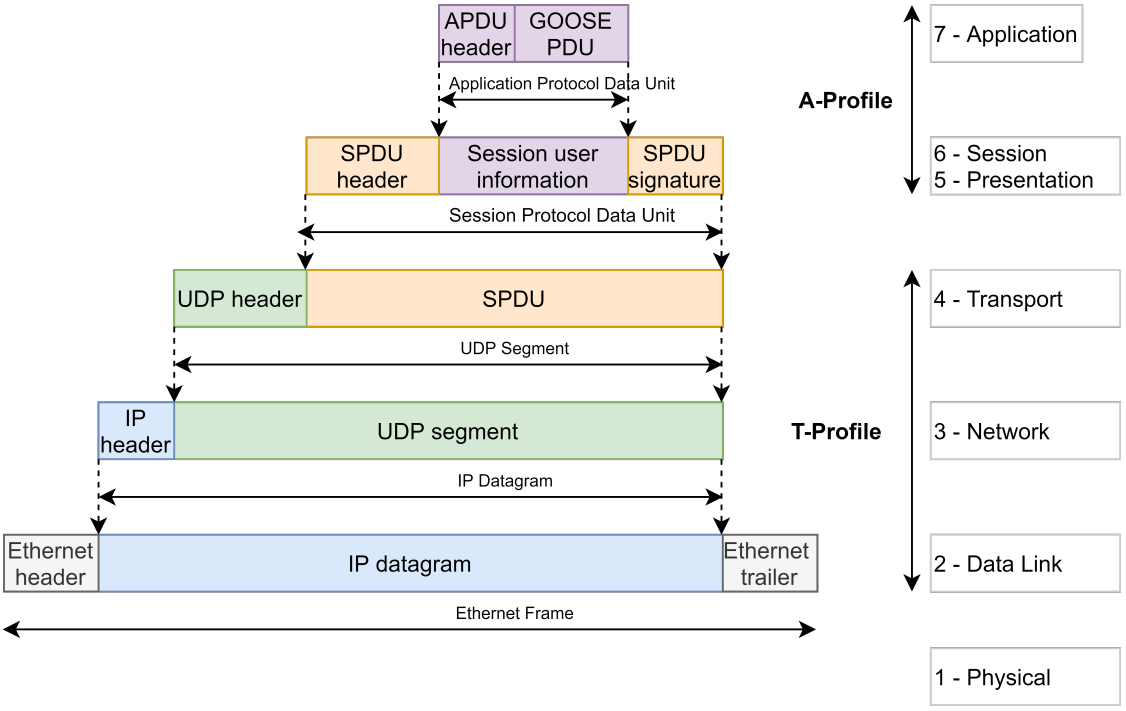

Figure 4.2: GOOSE message encapsulation

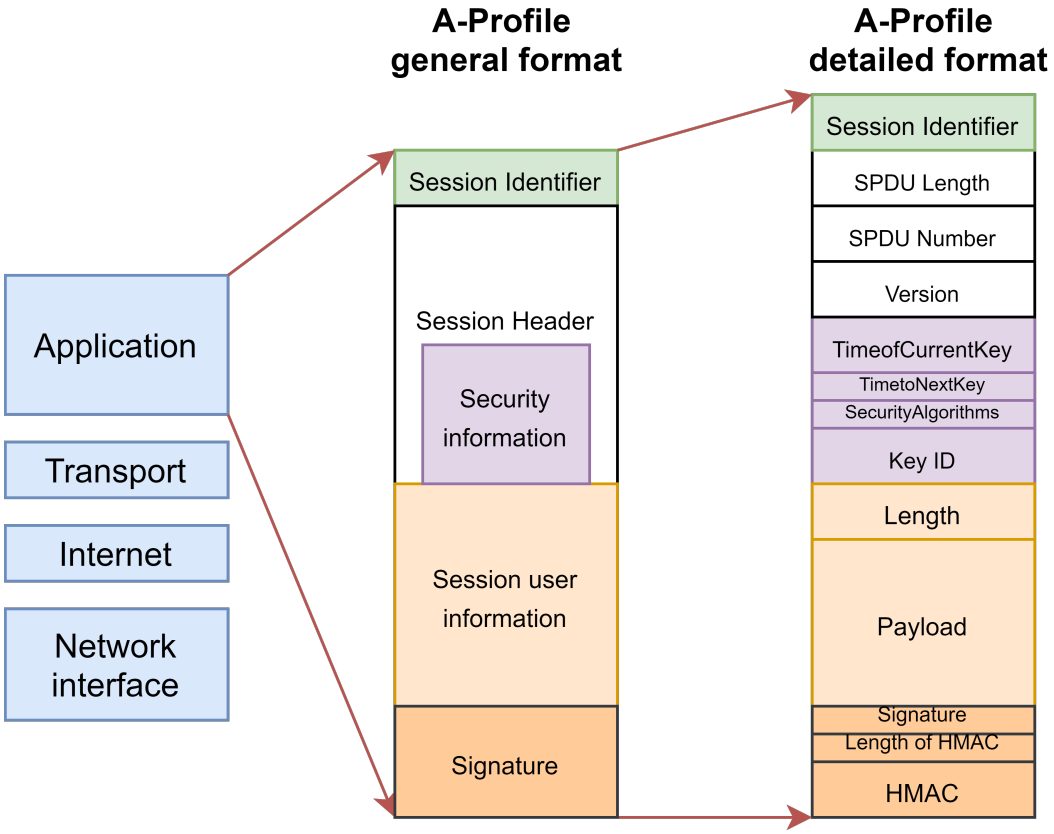

Figure 4.3: Structure of IEC TR 61850-90-5 session protocol [1]

The middle and left side of figure 4.3 shows the session protocol data unit (SPDU) in general and in detail. The SPDU starts with a Session Identifier (SI). The SI can have the following four values:

- **A0**: For tunneled GOOSE and SV
- **A1**: SPDUs that contain non-tunneled GOOSE Application Protocol Data Unit (APDUs)
- **A2**: SPDUs that contain non-tunneled SV APDUs
- **A3**: SPDUs that contain non-tunneled management (MNGT) APDUs

The Session Header contains the following attributes:

- **SPDU length**: Fixed size 4-byte word, max value of 65 517. Starts from SPDU number until the end of signature.
- **SPDU number**: Fixed size 4-byte integer used by subscriber to detect duplicate or out-of-order packet delivery.
- **SPDU Version**: Fixed size 2-byte integer representing the session protocol version number
- **Security Information**:
  - **TimeofCurrentkey**: fixed size 4-byte integer representing the time at which the current key for HMAC generation/encryption was first used

  - **TimetoNextKey**: fixed size 2-byte integer representing the time until the next key for HMAC generation/encryption is put to use
  - **SecurityAlgorithm**: fixed size 2-byte attribute indicating which cipher suit and algorithms are used to generate HMACs and encryption of the payload
  - **KeyID**: fixed size 4-byte attribute that refers to the key currently being used

The Session user information contains the following attributes:

- **Length**: Fixed size 4-byte integer with max value of 65399, used to represent the payload size.
- **Payload**: The payload structure depends on the PDU type (GOOSE, SV, Tunneled or MNGT). GOOSE and SV have the same structure. The first four attributes are the same, and they end with a GOOSE protocol data unit (GOOSE PDU) or Sampled Value data unit (SV PDU):
  - **Payload type**: Indicates the mapping of the payload, figure 4.4 shows the four different mapping
  - **Simulation**: Boolean value (True/False) used to indicate whether the payload is sent for test purposes or not.
  - **APPID**: Fixed size 2-byte used to distinguish the application association.
  - **APDU Length**: Fixed size 2-byte integer indicating the length of the GOOSE/SV PDU + the APDU length field itself.
  - **GOOSE/SV APDU**: The GOOSE PDU as defined in IEC 61850-8-1, or SV PDU as defined in IEC 61850-9-2

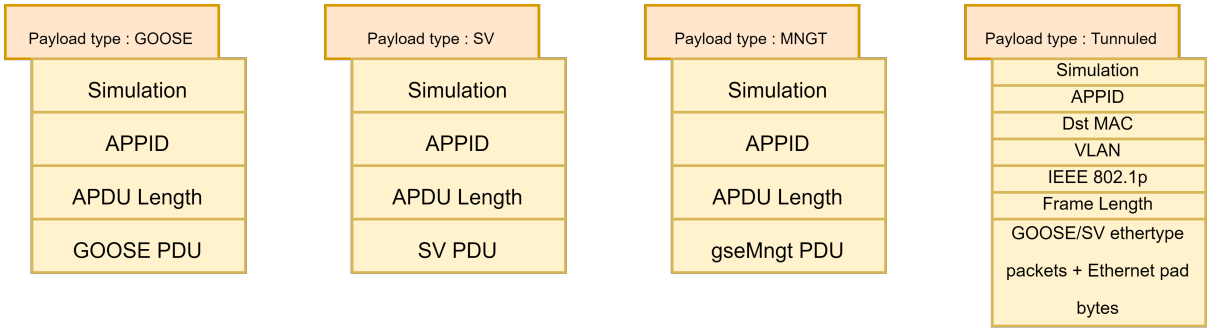

Figure 4.4: Payload types

The Signature contains the following attributes:

- **Signature**: 1-byte tag with value, specifying the underlying hash/HMAC function
- **length**: 1-byte integer indicating the length of the calculated HMAC
- **HMAC**: The calculated HMAC for the SPDU

4.2 Proposed Solution - Delayed integrity check

TR 90-5 offers mechanisms and functions to provide message integrity and authentication when transmitting synchrophasor information. But as it has been pointed out multiple times already, these solutions are not well suited for real-time critical communication. In this section, the alternative solution that this thesis suggests is presented in detail.

This thesis suggests a delayed integrity check that runs in parallel to the devices which exchange synchrophasor information. This is shown in figure 4.5.

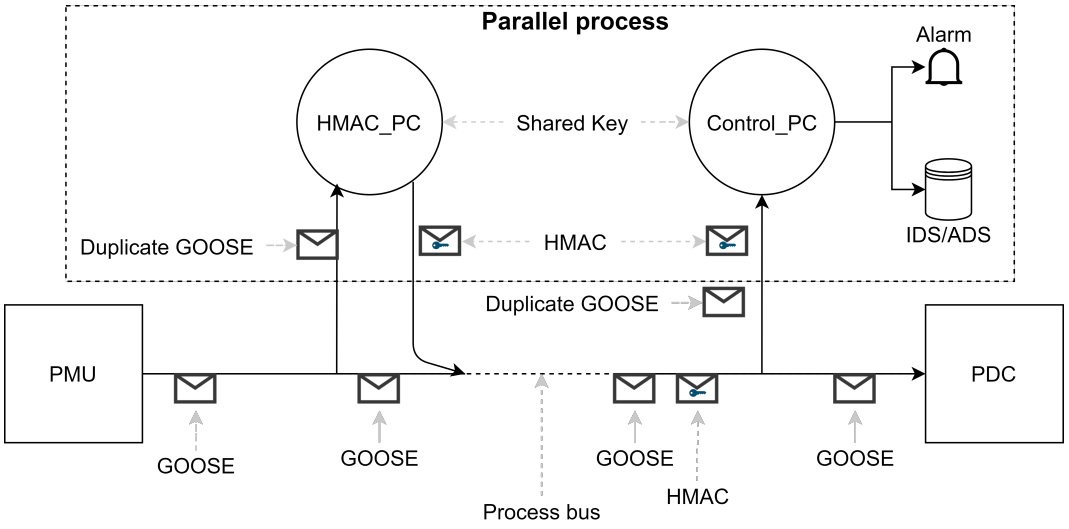

Figure 4.5: Proposed solution model [1]

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 (described in chapter 2) 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. However, this is an extremely simplified explanation, which leaves out many important details. The rest of this chapter is dedicated to explaining this solution in detail.

4.2.1 Solution design

Due to the limited time and resources at 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 thesis 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 to the same local network. One computer represents the PMU, one the HMAC-PC, and one the Control-PC. Since the purpose of the thesis 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 a 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. The code snippet below shows how this is performed:

```
# PMU.py
host = socket.gethostname()                          # Gets the IP-address of the PC
hmac_adr =  ("127.0.0.1", 8001)                      # IP address and port of HMAC-PC
controll_adr =  ("127.0.0.1", 9001)                  # Ip address and port of Control-pc
pmu= socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)
                                                     # Creates an UDP socket to send from
with open('SVpackets.txt') as f:                     # Opens txt file with GOOSE messages
    mylist = list(f)                                 # Creates of the text file
GOOSE_Array =[]                                      # Creates an empty list
for j in range(0, len(mylist)):                      # Iterates over the text file, adding
    if j % 18 == 0:                                  # each individual GOOSE message as
        packet = ""                                  # an element to GOOSE_Array
        for i in range(j, j+17):
            packet = packet + mylist[i]
        GOOSE_Array.append(packet)
length = len(GOOSE_Array)

for i in range(0, length):                               # For each element in GOOSE_Array:
    pmu.sendto(str.encode(GOOSE_Array[i]), hmac_adr)     # Sends a packet to hmac pc
    pmu.sendto(str.encode(GOOSE_Array[i]), controll_adr)# Sends a packet to controll pc
    time.sleep(0.015)                                    # max rate of PMU GOOSE messages

print("Program finished")
```

4.2.2 Key distribution - Diffie Hellman

Before the HMAC-PC and Control-PC can start receiving GOOSE messages, they must first establish the shared secret key that is used to generate the HMACs. To generate a shared secret key, an algorithm called Diffie-Hellman (DH) key exchange is used. DH is a public-key algorithm that enables two parties to securely establish a shared secret key over a public channel. This means that even if a third party were to eavesdrop on the exchange, they would only be able to obtain the public parameters used the create the secret key, not the key itself. The DH algorithm is shown in figure 4.6 below:

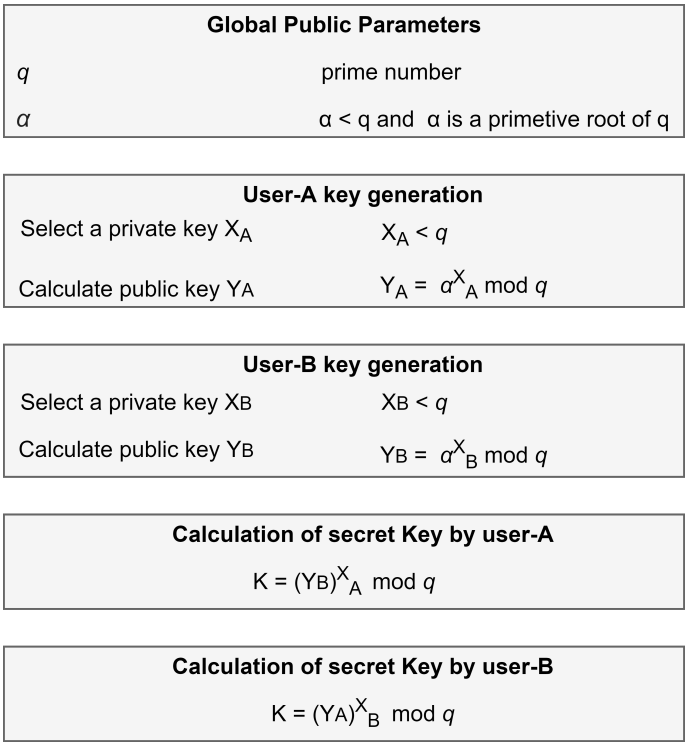

Figure 4.6: Diffe-Hellman key exchange algorithm [31]

Since X_A and X_B are private keys, an attacker would only be able to obtain q, α, Y_A, and Y_B by eavesdropping. To obtain one of the private keys, the attacker would have to calculate one of the following discrete logarithms: $X_B = dlog_{aq}(Y_B)$ or $X_A = dlog_{aq}(Y_A)$. The security of DH relies on the difficulty of calculating discrete logarithms. For large primes, the task is considered to be infeasible. Meanwhile, it is relatively easy to calculate exponentials modulo a prime [31]. An example calculation is shown in figure 4.7.

Global public parameters:
q = 2971
α = 3

Private keys:
X_A = 951 Private key A
X_B = 231 Private key B

Public keys:
A calculates $\mathbf{Y_A} = 3^{951} \bmod 2971 = 1032$
B calculates $\mathbf{Y_B} = 3^{231} \bmod 2971 = 131$

Public keys are exchanged
A calculates $\mathbf{K} = 131^{951} \bmod 2971 = \mathbf{1132}$
B calculates $\mathbf{K} = 1032^{231} \bmod 2971 = \mathbf{1132}$

Figure 4.7: Diffe-Hellman key exchange: practical example

The program running on HMAC-PC and Control-PC uses q = 886867 and $\alpha = 3$ as public parameters (set statically in the code). They both use a secure random number generator (within python) to generate private keys. The key exchange is initiated by the HMAC-PC. It starts by generating a random number and then calculates its own public key. This key is sent to the Control-PC. Control-PC responds by generating its own private key and then calculating a public key that is sent back to the HMAC-PC. Both computers now have all the information they need to generate a shared secret key. Figure 4.8 illustrates the key exchange between the two computers

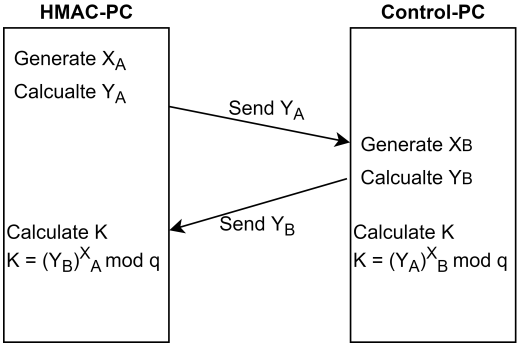

Figure 4.8: Diffe-Hellman key exchange messages

4.2.3 Key distribution - Blum Blum Shub

One weakness of DH is the strength of the key that it generates. Even when using high integers that produce keys with numerical seven-figure values, it is still trivial for an attacker to guess it by brute-force. To crack the key, an attacker would first have to intercept an HMAC and its corresponding GOOSE message. Then use brute-force to try and create an HMAC identical to the one he intercepted, with the intercepted GOOSE

message and random keys(integers) as input. When an HMAC matching the intercepted HMAC is generated, the attacker would successfully have guessed the correct key. Since both hashing and comparing are trivial and relatively quick operations, an attacker with an ordinary laptop could guess the key within minutes if a seven-figure numerical key is used.

This is why the key generated by the Diffie-Hellman algorithm is not used for HMAC generation, but rather as a "shared secret seed". This seed is used to initialize a cryptographically secure pseudorandom bit generator (CSPRBG) called Blum Blum Shub (BBS). BBS is named after its inventors (Lenore Blum, Manuel Blum, and Michael Shub) and has perhaps the strongest public proof of its cryptographic strength [31]. A CSPRBG is a deterministic algorithm that produces a "seemingly random" sequence of bits. Deterministic means that it will produce the exact same sequence of bits, given the same input. Seemingly random means that the sequence should be indistinguishable from a truly random sequence. For such an algorithm to be considered cryptographically secure, it should be infeasible to determine the seed used or the next bit in the sequence, given a sequence of bits (of any length).

Figure 4.9 shows the global parameters and the algorithm itself. BBS uses two public parameters, q, and p. These must be two large prime numbers that both have a remainder of 3 when divided by 4. The notation for this is shown in the figure below.

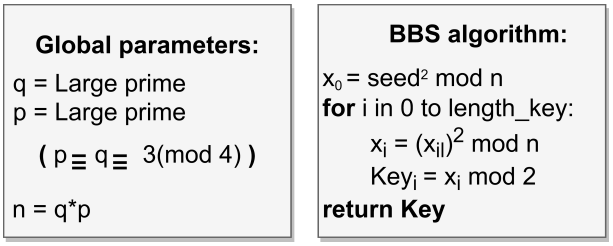

Figure 4.9: Blum Blum Shub parameters and algorithm

For each iteration of the for loop, a "random" bit is appended to the key. The loop is run as many times as there are bits in the key. In the solution, the key length have been set to 264 and the global parameters have been set to $\mathbf{q} = 32452843$ and $\mathbf{p} = 15485863$. The code snippet below shows how this has been implemented in python. HMAC-PC and Control-PC use the same function and parameters.

```
def BBS(seed, key_length):
    q = 32452843
    p = 15485863
    n = q*p                              # q mod(3) == p mod(3)
    key = ''
    for i in range(0, key_length):       # BBS formula realization
        seed = (seed**2)%n
```

```
        bit = seed & 1                  # Bit selection method; keeping only the LSB
        key += str(bit)
    key = int(key, 2)                   # Transforming the binary key to into an integer
    return key
```

4.2.4 HMAC generation

An 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 an 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 [32]:

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

- 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 an 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 turn opens a loophole that 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 an 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 to high, an attacker could potentially send enough malicious GOOSE messages to trick the control mechanism to make a devastating ill-informed decision before the HMAC is sent and the attack is detected.

The solution provided in this thesis 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, the shared secret key generated in by the Diffie-Hellman key exchange is used as a seed 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, a new random value between 1-250 is selected. The seed is updated using the BBS function with itself as input so that the same value is not picked again. The code snippet below shows how this is performed

```
random.seed(RandomKey)              # Initial RandomKey = Key from DH-exchange
randval = random.randrange(1,250)   # Random value between 1-250
if ID % randval == 0:               # Selectes messages with ID dividable with randval
    single_HMAC = hmac.new(bytes(key), b'', hashlib.sha256,)    # Create/Reset single-hmac
    single_HMAC.update(msg.encode("utf-8"))                     # Calculate new single-hmac
    RandomKey = int(BBS(RandomKey, 10))                         # Updates seed, using BBS
```

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 receives 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 comparing the two HMACs, the string is reset to nothing.

4.2.5 Alarm generation

The whole point of this solution is to detect if the integrity between the communicating parties has been compromised. How to react if this is detected is not within the scope of this thesis. 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%

Chapter 5

Experimental Evaluation

In this chapter, the delayed integrity will be demonstrated in practice. The first part of the experiment will demonstrate how the program runs under normal conditions. The second part will show how it manages to detect malicious GOOSE messages injected into the network. In addition, a few simple tests will be used to estimate the time it takes to perform the integrity check.

5.1 Experimental Setup and Data Set

As mentioned in chapter 3, the main functionality of the delayed integrity check developed in this thesis lies within the three individual python scripts PMU.py, HMAC-PC.py, and Control-PC.py. The experimental setup is shown in figure 5.1. When testing the solution, each of the scripts is run on an individual laptop. The laptops are all connected to the same local network via a regular switch.

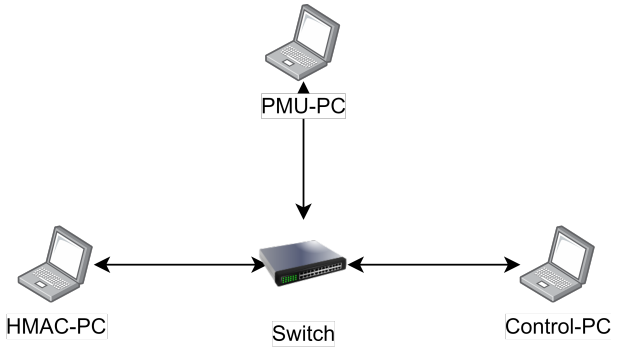

Figure 5.1: Experimental setup

The data set used is a Wireshark capture of real GOOSE traffic (.pcap file), captured at Statnett's digital test-substation in Oslo. An example of a captured GOOSE message is

shown in figure 5.2. The capture contains approximately 641.000 captured packets, where 345.000 of them are GOOSE messages. PMU.py filters out these GOOSE messages and converts the data set into an array list, where each element is a real GOOSE message. Each individual GOOSE message is stripped of the headers from the transport layer and underlying protocols on the communication stack (marked with black in the figure below). What is left in the array list is the A-profile (ref figure 4.2), which is shown inside the blue part of figure 5.2

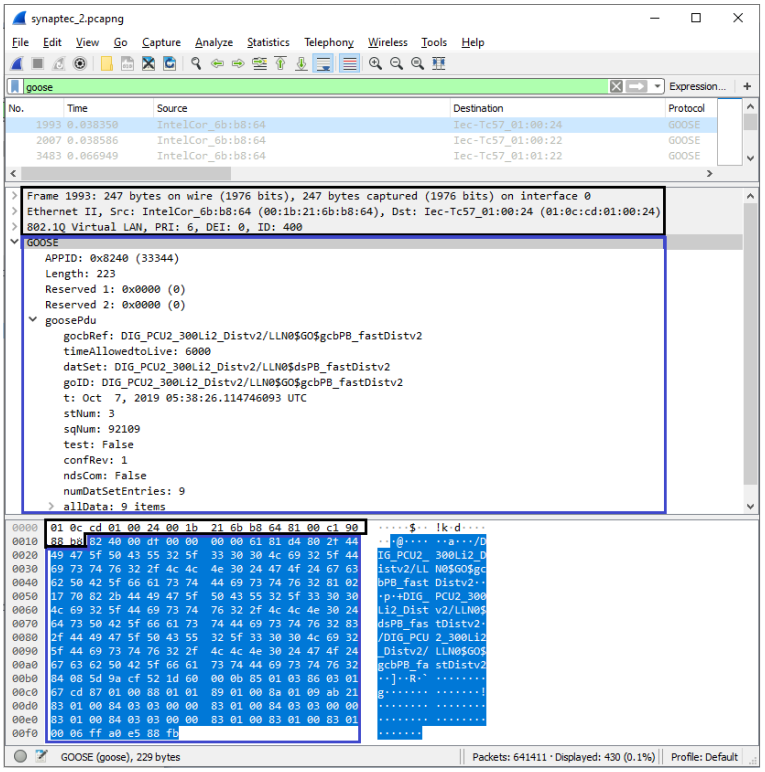

Figure 5.2: Wireshark capture of process bus traffic

The array list is iterated through, and for each iteration, a GOOSE message is sent from the PMU to the HMAC and Control PC. In a real digital substation, the PMU would multicast these GOOSE messages out on the process bus. This is not the case in this experiment. Since the goal of this thesis is to serve as a proof of concept, the GOOSE messages are simply unicast to the HMAC and Control computer. Their IP addresses have been manually configured in PMU.py.

When designing a security solution, one has to assume that the adversary is capable of everything. If there is a flaw in the design, an attacker will find it and exploit it. Which is why every possible scenario has to be mitigated. The worst-case scenario in the context of the delayed integrity check would be if an attacker were to gain access to the process bus and manages to disguise itself as a legitimate PMU. In other words a combination of a spoofing and packet injection attack. If the HMAC and Control PC simply picked

up GOOSE messages that the PMU multicast out on the process bus, they would not be able to detect this type of attack. This is why placement of the HMAC-PC is of critical importance. In a real-world digital substation, the HMAC-PC would be directly connected to the PMU and would only listen for GOOSE messages on this interface. That way if an attacker were to spoof the PMU and inject malicious traffic into the process bus, this traffic would never reach the HMAC-PC.

When testing how the solution performs under attack, the same setup as in figure 5.1 is used, but a few modifications are made to PMU.py. In this scenario, we assume that the HMAC-PC is directly connected to the PMU, and don't listen to traffic on the process bus. It is also assumed that the adversary has managed to spoof the PMU, and injected malicious traffic into the process buss. This traffic would be indistinguishable from legitimate traffic produced by the PMU.

The modifications made to PMU.py to simulate this attack are quite simple. Instead of sending each GOOSE message to the HMAC and control PC, some of the messages are just sent to the Control PC. This would be the case if an attacker were to inject traffic into the process bus. In a real digital substation, the Control PC would be placed as close to the PDC as possible (topology-vise) and pick up all the GOOSE messages sent by the PMU.

5.2 Experimental Results

5.2.1 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. Figure **??** shows a Wireshark capture of this exchange. An attacker that eavesdrops on the key-exchange can obtain all the information shown in the two Wireshark windows, but as explained in chapter 4.2.2 it is still considered to be infeasible to calculate the shared session key (seed). The shared secret key (seed) and both Diffie-Hellman public keys are shown in the terminal output, this is just for the illustration purpose of this figure and is not shown otherwise.

After the key exchange, the PMU script is run. Figure 5.4 shows the terminal output from each script when the program is run as normal (no attack).

As can be seen in figure 5.4, 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 are 3.18 single-HMACs are sent between each buffer-HMAC. The number 3.18

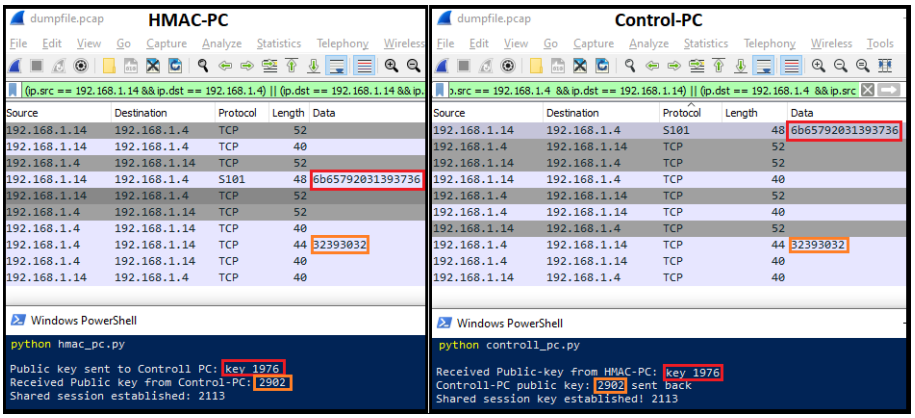

Figure 5.3: key-exchange Wireshark capture and terminal output

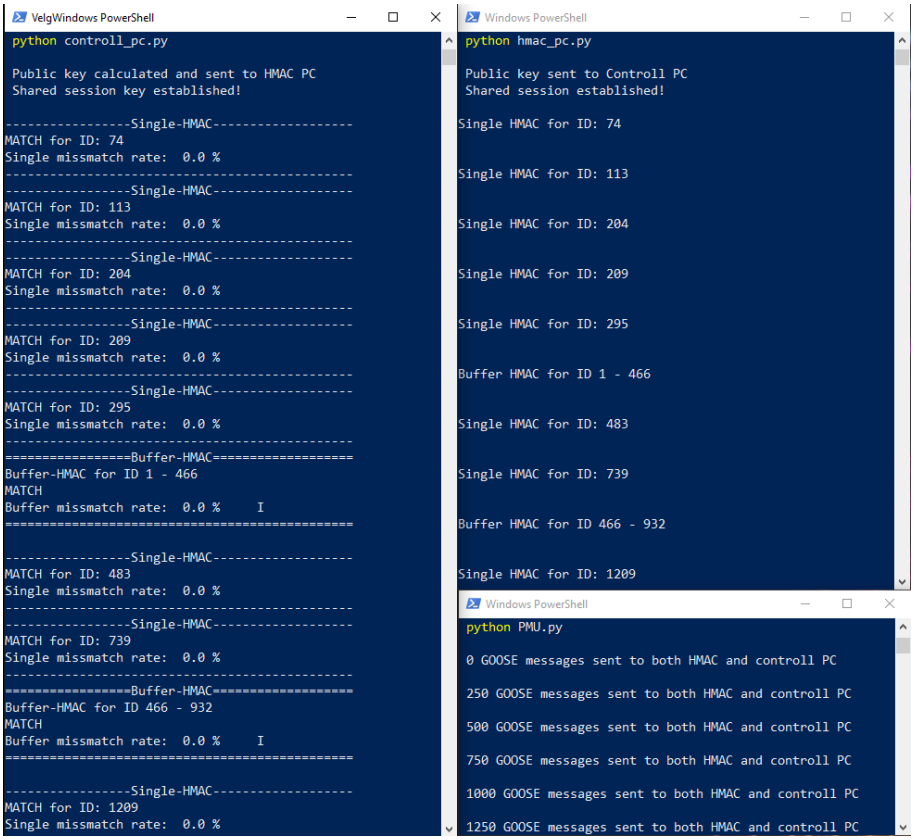

Figure 5.4: Terminal output

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 in figure 5.4 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 SV and PTP making up the rest. The IDs are given sequentially to every packet, independent of protocol.

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. Figure 5.5 shows the terminal output after running a full simulation.

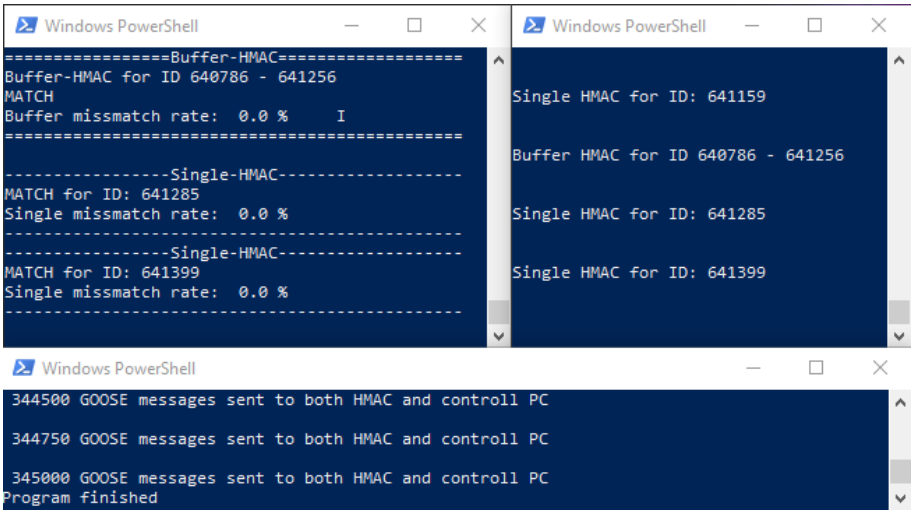

Figure 5.5: Terminal output for full simulation

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. This issue will be discussed further in chapter 5.

5.2.2 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. Figure 5.6 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.

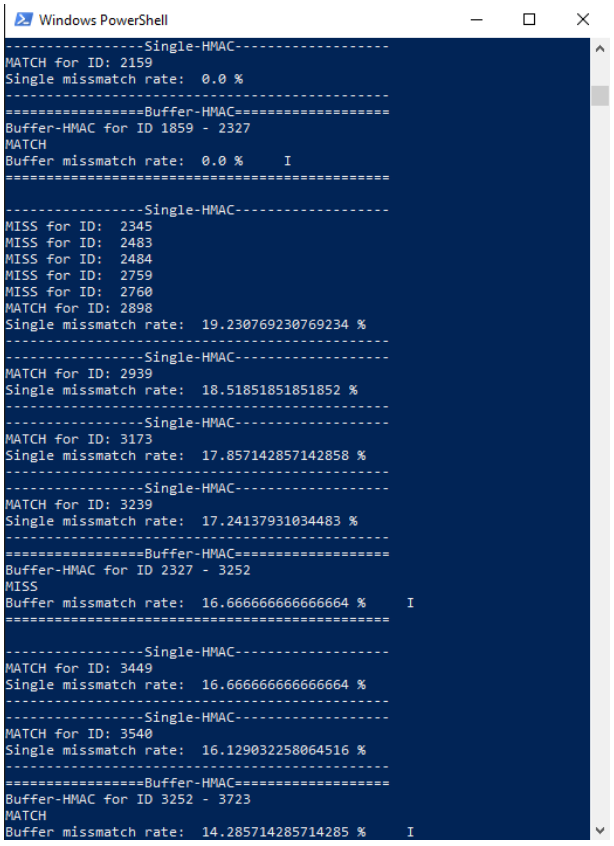

Figure 5.6: Attack 1 output: Injected stream

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. Preventing 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), this will be discussed further in chapter 5.

In the second attack, GOOSE messages will be injected at random, instead of as a sequential stream of messages. PMU.py has 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. The modifications made to PMU.py are shown in the code snippet below:

```
for i in range(0, length):                          # For each GOOSE msg DO:
    random.seed(i)
    randval = random.randrange(0,100)               # Select a random value 0-100
    if randval <= 25:                               # If the value <=25:
        pmu.sendto(GOOSE_Array[i], control_adr) # Sends a packet ONLY to control pc
    else:                                           # 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
```

Figure 5.7 shows the output generated during this simulated attack. After running the simulation for 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.

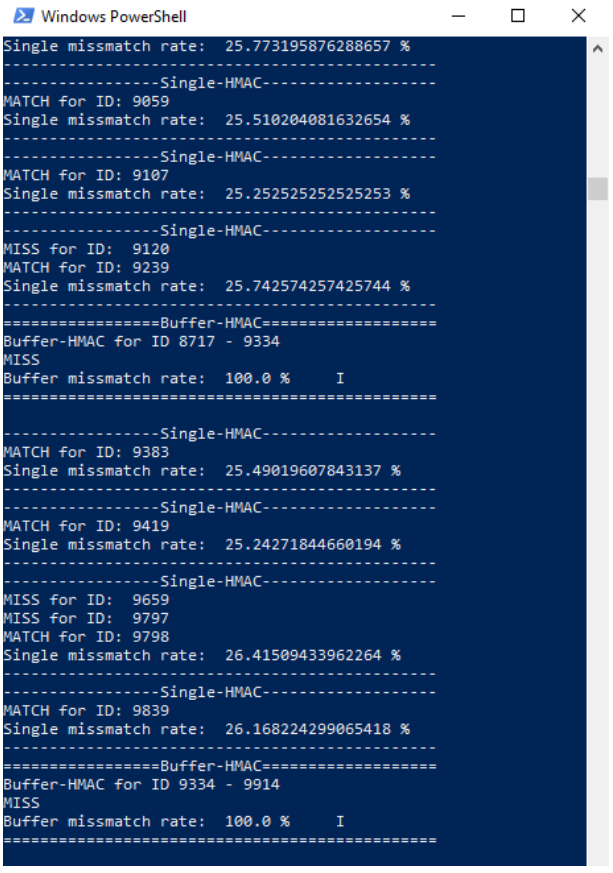

Figure 5.7: Attack 2 output: Random injections

5.2.3 Delay

Now that the integrity check has 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 that would cause damage to the substation, 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. the code snippet below shows this test.

```
t = time.time()
for i in range(0, length):
    single_HMAC = hmac.new(bytes(key), b'', hashlib.sha256,)
    single_HMAC.update(GOOSE_Array[i].encode("utf-8"))
elapsed = time.time() - t
print("Average time to create a HMAC: ", elapsed/length)
```

This test creates an 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 a 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 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 has been carried out over Ethernet cables on a LAN with no other traffic, the delay in this experiment is not representative. 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" [33] is used. The researchers found that the delay on their process bus was minimum 15.2 μ-sec, average 16.5 μ-sec and maximum 17.7 μ-sec when sending SV messages from a merging-unit to a PMU. Since the message structure and protocol for both SV and GOOSE are so 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 an 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 are received at a PDC/application, to the time a mismatch is detected is shown in figure 5.8

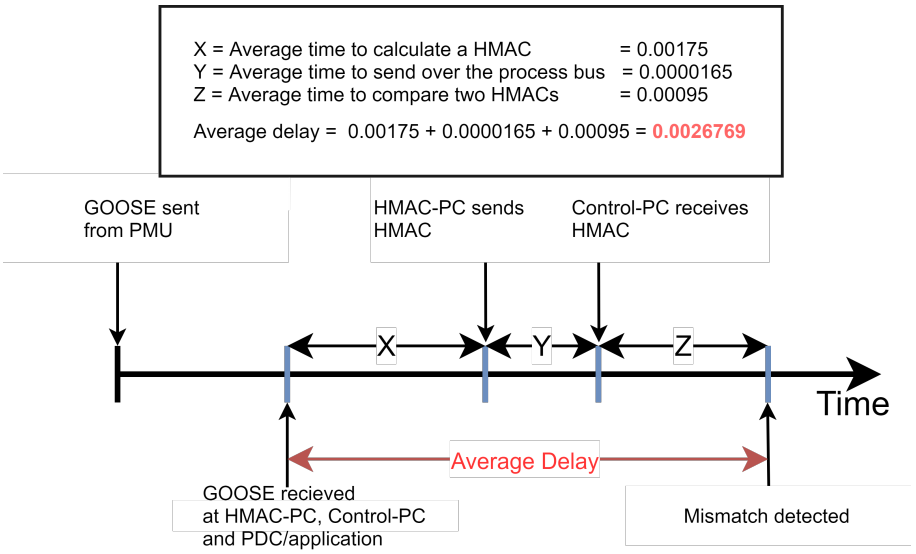

Figure 5.8: Average delay

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 detects it.

There is currently very little information available on exactly how many GOOSE messages are needed for a control application to make a decision that (if incorrect) would cause damage to the substation transformer. However, transformers gradually adjusted based on synchrophasor measurements, and protection mechanisms prevent them from adjusting to fast and drastically. Which is why a single malicious measurement is not enough to cause any damage. PMU researchers at Statnett suggest that it would take **more** than a couple of seconds for a transform to adjust enough to damage itself if subjected to incorrect synchrophasor measurements. Thus with the current rate of single/buffer HMAC generation, an attacker would not be able to send enough malicious messages GOOSE message to cause any damage before being detected (even in the worst-case scenario described in the section above).

Chapter 6

Discussion

In this chapter the following topics will be discussed: packet loss, false-positives mismatches, anomaly based intrusion detection systems (ADS), and GOOSE re-transmission

6.1 HMAC generation

A viable option that has been barley been mentioned, is to generate HMACs for every single GOOSE message and use a separate VLAN (or separate network) 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 a real process bus. It will be recommended as further work for this thesis to test this. 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 the one that this thesis suggests as the best option.

Using an entirely different network (or even wireless 5G) for the HMAC traffic is an option that would generation of an HMAC for each GOOSE individual message, without introducing any delay to the GOOSE traffic. This would require a more network infrastructure and increase the overall cost, but could definitely be a valid option if the owner of the substation is willing to take the increased cost.

6.2 GOOSE re-transmission

In chapter 2 it was mentioned that GOOSE messages are re-transmitted until a new event occurs and that 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 by keeping track of the ID of the last seen GOOSE message. For each message that has a state number that indicated that the message is a re-transmission, the ID in the re-transmitted message is compared 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 an 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) - This will be suggested as future work in chapter 7.

6.3 Packet loss

In chapter 5.2.1, 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 messages being sent on the network.

Packet loss is usually caused by network congestion, problems with network hardware, software bugs, and overloaded network devices. According to Statnett, the network

infrastructure on their digital substation pilot project have the capability to handle more than twice the traffic load of normal operating conditions. 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 therefore 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.

6.4 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 taught 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.

Chapter 7

Conclusion and Future Directions

7.1 Conclusion

In this thesis, a solution for a delayed integrity check for IEC 61850 GOOSE communication was presented. The solution offered a way to ensure integrity, without introducing any extra delay or overhead to the GOOSE traffic. An introduction to digital substations and the IEC 61850 [21] standard, as well as a review of the existing solution within IEC TR 61850-90-5 [1] were given. Security objectives and potential attacks against synchrophasor systems were also presented. It was 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. This thesis 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 (running a different python script each) 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.

7.2 Future work

The goal of the program that was developed in this thesis 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. Further investigation into the optimal gap between single HMACs, and the number of messages buffered pr buffer-HMAC is also recommended.

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 as a part of 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.

List of Figures

Bibliography

[1] 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.

[2] Hassan Haes Alhelou, Mohamad Esmail Hamedani Golshan, Takawira Njenda, and Pierluigi Siano. A survey on power system blackout and cascading events: Research motivations and challenges. *Energies*, 12, 02 2019. doi: 10.3390/en12040682.

[3] Stein Ingebrigtsen Harald Hole Dietrichson Rannveig S. J. Løken Nargis Hurzuk, Svein Losnedal. Digitale stasjoner i transmisjonsnettet. page 12, 03 2018.

[4] Circuitglobe. Electrical Substation Definition. https://circuitglobe.com/electrical-substation.html.

[5] S.V. Kulkarni and S.A. Khaparde. *Transformer Engineering: Design, Technology, and Diagnostics, Second Edition*. Taylor & Francis, 2012. ISBN 9781439853771. URL https://books.google.no/books?id=1kzWPb4ok2EC.

[6] Edvard Csanyi. Advantages Of IEC 61850. https://electrical-engineering-portal.com/advantages-of-iec-61850, .

[7] ABB brouchure. We are bridging the gap - Enabling Digital Substations, p13. https://library.e.abb.com/public/c3f8221f76db46b38d9ae6f8ec6475b2/ABB_Digital%20Substation%20Brochure_1.0.4.pdf.

[8] Grid ALSTOM. Network protection & automation guide, edition may 2011.

[9] Edvard Csanyi. IED (Intelligent Electronic Device) advanced functions that make our life better. https://electrical-engineering-portal.com/ied-intelligent-electronic-device-advanced-functions, .

[10] Rafiullah Khan, Kieran Mclaughlin, David Laverty, and Sakir Sezer. IEEE c37.118-2 synchrophasor communication framework: Overview, cyber vulnerabilities analysis and performance evaluation. 01 2016. doi: 10.5220/0005745001670178.

[11] SmartGridGov. Smart Grid Definitions of Functions pdf, p2. https://www.smartgrid.gov/files/definition_of_functions.pdf.

[12] IEEE standard for synchrophasor data transfer for power systems. *IEEE Std C37.118.2-2011 (Revision of IEEE Std C37.118-2005)*, pages 1–53, Dec 2011. ISSN null. doi: 10.1109/IEEESTD.2011.6111222.

[13] IEEE. Ieee standard for synchrophasor measurements for power systems. *IEEE Std C37.118-2005*, 2005. ISSN null.

[14] IEEE standard for synchrophasor measurements for power systems. *IEEE Std C37.118.1-2011 (Revision of IEEE Std C37.118-2005)*, pages 1–61, Dec 2011. ISSN null. doi: 10.1109/IEEESTD.2011.6111219.

[15] Victorya University. Implementation of IEC61850 in a Substation Environment. https://electrical-engineering-portal.com/res/Power-Substation-Automation-and-Communication-IEC-61850.pdf.

[16] Hangtian Lei, C. Singh, and Alex Sprintson. Reliability modeling and analysis of iec 61850 based substation protection systems. *Smart Grid, IEEE Transactions on*, 5:2194–2202, 09 2014. doi: 10.1109/TSG.2014.2314616.

[17] ISO 9506. Industrial automation systems, manufacturing message specification. part 1, iso iso standard iso 9506-1:2003. 2003.

[18] IEC. Communication networks and systems for power utility automation - part 8-1: Specific communication service mapping (scsm) - mappings to mms (iso 9506-1 and iso 9506-2) and to iso/iec 8802-3. .

[19] Christoph Brunner. Peer-to-peer communication in iec 61850. http://www.pacw.org, 11 2016.

[20] IEC. Communication networks and systems for power utility automation - part 9-2: Specific communication service mapping (scsm) - sampled values over iso/iec 8802-3. .

[21] 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. pages 1–148, .

[22] Electropedia. International Electrotechnical Commission Glossary - Abstract Communication Service Interface (ACSI). http://www.electropedia.org/, .

[23] Electropedia. International Electrotechnical Commission Glossary - Specific Communication Service Mapping SCSM. http://www.electropedia.org/, .

[24] Defense Use Case. Analysis of the cyber attack on the ukrainian power grid. *Electricity Information Sharing and Analysis Center (E-ISAC)*, 388, 2016.

[25] C. Beasley, G. K. Venayagamoorthy, and R. Brooks. Cyber security evaluation of synchrophasors in a power system. In *2014 Clemson University Power Systems Conference*, pages 1–5, March 2014. doi: 10.1109/PSC.2014.6808100.

[26] M. D. Hadley, J.B. McBride, T.W. Edgar, L.R. O'Neil, and J.D. Johnson. Securing wide area measurement systems. Technical report, 2007. URL https://www.energy.gov/sites/prod/files/oeprod/DocumentsandMedia/8-Securing_WAMS.pdf.

[27] Yan Lu, Mohsen Jafari, Paul Skare, and Kenneth Rohde. An integrated security system of protecting smart grid against cyber attacks. pages 1 – 7, 02 2010. doi: 10.1109/ISGT.2010.5434767.

[28] Rafiullah Khan, Kieran McLaughlin, David Laverty, and Sakir Sezer. Analysis of ieee c37. 118 and iec 61850-90-5 synchrophasor communication frameworks. In *2016 IEEE power and energy society general meeting (PESGM)*, pages 1–5. IEEE, 2016.

[29] Seyed Reza Firouzi, Luigi Vanfretti, Albert Ruiz-Alvarez, Hossein Hooshyar, and Farhan Mahmood. Interpreting and implementing iec 61850-90-5 routed-sampled value and routed-goose protocols for ieee c37. 118.2 compliant wide-area synchrophasor data transfer. *Electric power systems research*, 144:255–267, 2017.

[30] Martin Gilje Jaatun and Marie E. G. Moe. PMU/PDC security considerations using IEC TR 61850-90-5. SINTEF memo, 2019.

[31] William Stallings. *Cryptography and Network Security: Principles and Practice*. Prentice Hall Press, USA, 5th edition, 2010. ISBN 0136097049.

[32] Dr. Hugo Krawczyk, Mihir Bellare, and Ran Canetti. HMAC: Keyed-Hashing for Message Authentication. RFC 2104, February 1997. URL https://rfc-editor.org/rfc/rfc2104.txt.

[33] Huan Yang, Liang Cheng, and Xiaoguang Ma. Analyzing worst-case delay performance of iec 61850-9-2 process bus networks using measurements and network calculus. pages 12–22, 05 2017. doi: 10.1145/3077839.3077856.

Appendix A

Program files

Attached in delayed-integrity-check.7z are the three following python scripts and data-set:

- PMU.py 
- controll_pc.py 
- HMAC_pc.py 
- packets.txt 

The following public GitHub respiratory (Delayed-integrity-check) contains the same files, as well a a "READ ME" file which explains how to run the program.

Appendix B

Published paper

(The following paper is not the final draft)

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.**

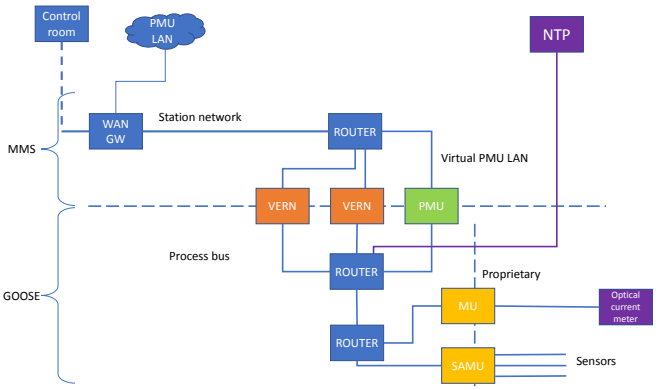

Fig. 1. Example digital substation with PMU

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 keypairs 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,

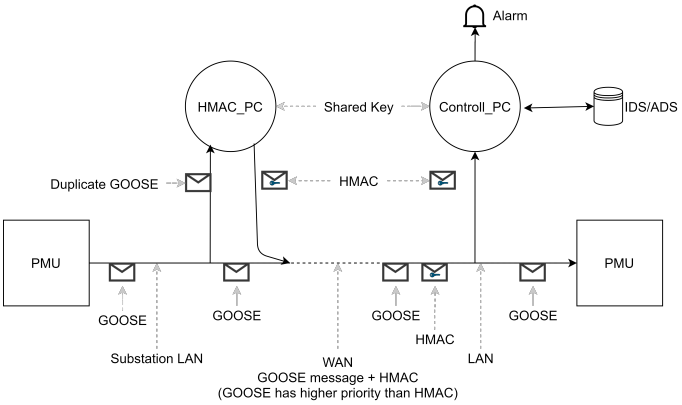

Fig. 2. Delayed integrity check concept

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) snaps up 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, the shared secret key generated by the 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, a new random value between 1-250 is selected. Then the seed is updated using the 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 SV and 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.

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.

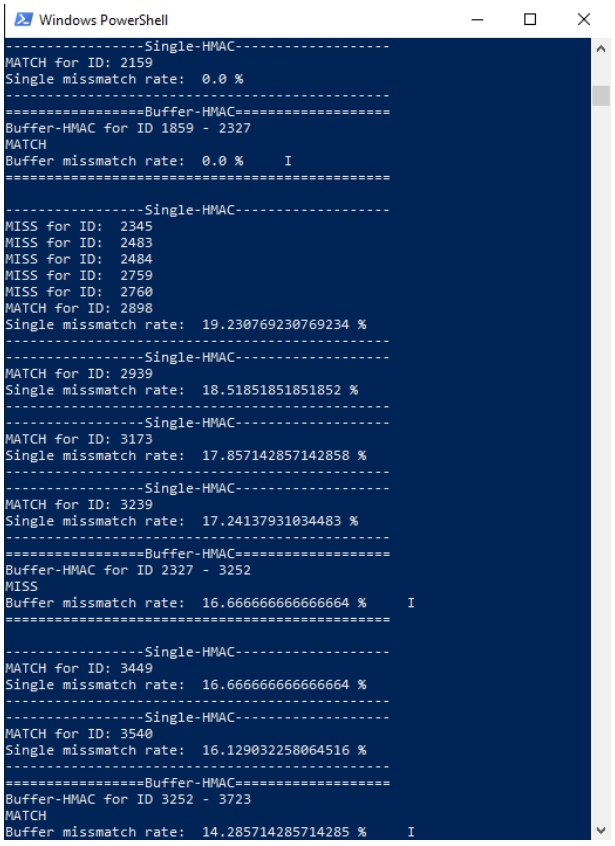

Fig. 3. Attack 1 output: Injected stream

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 μ-sec, average 16.5 μ-sec and maximum 17.7 μ-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

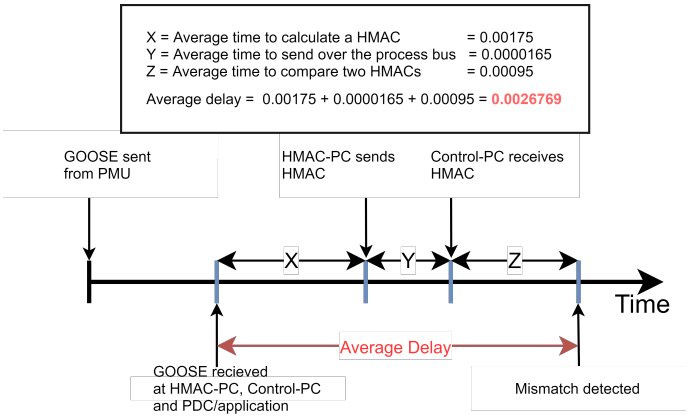

Fig. 4. Average delay

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 are 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, 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. H. et al, "Securing wide area measurment systems pacific northwest national laboratory, report nr: PNNL-17116.2007," Tech. Rep.

[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. E. G. M. Martin Gilje Jaatun, "Pmu/pdc security considerations using iec tr 61850-90-5," 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.