Cryptographic Solutions for Cyber-Physical System Security

Chenglu Jin
University of Connecticut - Storrs, chenglu.jin@uconn.edu

Cryptographic Solutions for Cyber-Physical System Security

Chenglu Jin, Ph.D.

University of Connecticut, 2019

ABSTRACT

Since the introduction of Stuxnet, security research of industrial control systems, or cyber-physical systems (CPSs) in general, has become a rapidly growing area. There is a widespread belief that solutions based on cryptographic primitives are generally considered too computationally expensive to realize security properties for CPS in practice.

In this dissertation, we will show how to efficiently leverage the limited computational power and storage on CPS devices to secure a CPS under the attacks initiated from sensors, controllers, and networks using cryptographic methods. More specifically, we will present an intrusion-tolerant and privacy-preserving sensor fusion scheme, a lightweight intrusion detection system for industrial control systems, and a multi-factor authenticated key exchange protocol based on historical data.

Cryptographic Solutions for Cyber-Physical System Security

Chenglu Jin

M.S., New York University, 2014

B.S., Xidian University, 2012

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Connecticut

2019

APPROVAL PAGE

Doctor of Philosophy Dissertation

Cryptographic Solutions for Cyber-Physical System Security

Presented by

Chenglu Jin, B.S., M.S.

Major Advisor ______________________________
Marten van Dijk

Associate Advisor ______________________________
John Chandy

Associate Advisor ______________________________
Benjamin Fuller

University of Connecticut

2019

ACKNOWLEDGMENTS

First of all, I would like to express my deepest appreciation to my advisor Prof. Marten van Dijk, who taught me to be fearless in my research and my life, and gave me so much academic freedom; otherwise, it would not be possible for me to dive into the area of cyber-physical system security. Also, many thanks to Marten for the insightful discussions and detailed guidance during my Ph.D. study. I am so fortunate to have you as my Ph.D. advisor for five years.

Thanks should also go to the rest of my Ph.D. committee, Prof. John Chandy and Prof. Benjamin Fuller. You gave me so much invaluable feedback and encouragement, not only for my dissertation writing but also for my Ph.D. study.

I would like to extend my sincere thanks to Prof. Jianying Zhou. I had a great time studying in Singapore for three months. This experience helped me quickly get into the area of industrial control system security. I gained a lot of real-world experiences and great friends over there: Dr. Zheng Yang, Sridhar Adepu, Chuadhry Mujeeb Ahmed, John Henry Castellanos Alvarado, and Andrei Bytes. Similarly, I would like to thank Dr. Junfeng Fan, who accepted me twice as a summer intern, for his belief in my work and research abilities.

Special thanks to Prof. Srini Devadas and Dr. Ulrich Rührmair for their valuable advice and support. Their passion for research always inspires me to go for the next challenge.

I am also grateful to all my fellow lab mates: Dr. Phuong Ha Nguyen, Dr.

Haibin Zhang, Dr. Syed Kamran Haider, Dr. Hoda Maleki, Reza Rahaeimehr, Saeed Valizadeh, Tara John, Kaleel Mahmood, Syed Rizwan Haider, and Deniz Gurevin. I am such a lucky man; I have collaborated with everyone in our lab on at least one project. None of them can be possible without you. I will never forget the difficult problems we struggled with and all the fun we had together.

Last but certainly not least, I have to express my sincere thanks to my parents and my wife for the continuous unconditional support and unparalleled love.

Contents

Chapter 1

Introduction

Cyber-Physical Systems (CPSs), like water treatment systems, power grids, and nuclear plants, are critical for the daily life of millions of people. However, the security of this kind of system is always an afterthought, which opens a tremendous attacking surface on CPSs for malicious adversaries [65, 176, 99, 24]. In recent years, we observed many security incidents in various cyber-physical systems [65, 104, 149, 119, 155, 99]. For example, Stuxnet, as a computer worm, was designed specifically to tamper with the control programs on Siemens Programmable Logic Controllers (PLCs) stealthily[65, 145]. It is reported that the infected PLCs in nuclear plants will operate nuclear centrifuges at a different rotational speed from expected. This modification can lead to physical damage on the nuclear centrifuges over time, and it is believed that one-fifth of the nuclear centrifuges in Iran were ruined due to Stuxnet. Recently, the attacks on CPSs are reported more frequently: Stuxnet, which was uncovered in 2010, attacked nuclear plants in Iran [65], Industroyer caused the Ukrainian blackout in 2016 [128], Triton almost triggered an explosion at petrochemical plants

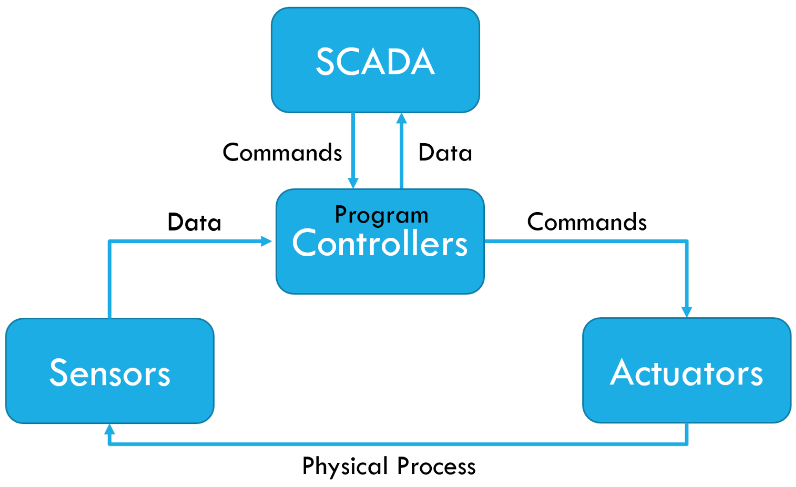


FIGURE 1.1: A general model of a cyber-physical system.

in Saudi Arabia in 2017 [99]. These incidents demonstrate the urgent demand of securing cyber-physical systems, especially critical infrastructures, against various attacks, and this dissertation serves as one important step toward building secure cyber-physical systems.

1.1 Cyber-Physical Systems

Cyber-physical systems, as the name indicates, are a special type of systems that interact with both cyber and physical worlds. Figure 1.1 depicts a general model of a cyber-physical system. It consists of three main components: sensors, controllers, and actuators. Sensors are measuring physical quantities at real-time and provide the measured data to the controllers, so they serve as a gateway from the physical world to the cyber world. After receiving sensor data, the controllers will make control decisions according to its program, internal states, and the sensor data. Then commands will be issued to the actuators, and the actuators will react to these commands and influence the physical process. This converts the information in the cyber world back to the physical world, and the effects will eventually be captured by the sensors. This completes a typical control loop of a cyber-physical system.

In the context of industrial control systems or process control systems, like power grids or nuclear plants, there is usually a supervisory control and data acquisition (SCADA) system sitting at the back end, as shown in Figure 1.1. It interacts with controllers, and monitors (at a high level) the whole physical process, issues commands to the controllers, or update the programs on the controllers. SCADA system is usually composed of the workstations used by engineers or operators, networking devices, and human-machine interfaces, so it is vulnerable to the cyber attacks on general computing systems, like computer viruses, as well.

1.2 The Scope of This Dissertation

In this dissertation, we focus on the security issues in industrial control systems as examples of cyber-physical systems, although some of our solutions can be extended to more general cyber-physical systems.

1.3 Industrial Control Systems Hierarchy

To better understand the security issues in industrial control systems, we have to introduce some background on how an industrial control system is structured from a controller's perspective. The system hierarchy is depicted in Figure 1.2.

In general, from a controller's perspective, an industrial control system has five

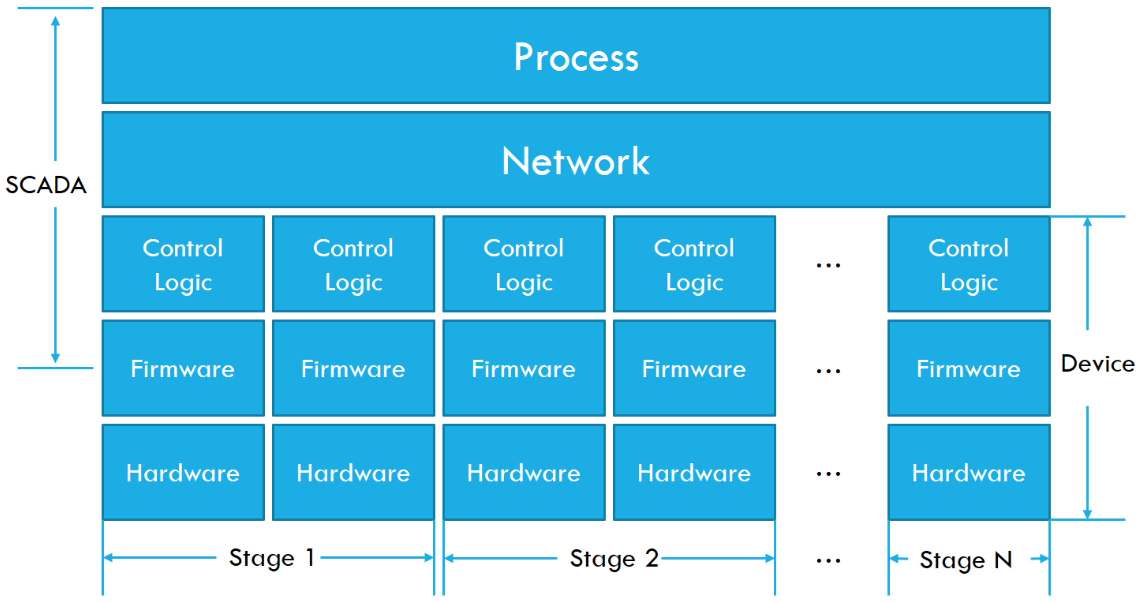


FIGURE 1.2: The system hierarchy of an industrial control system.

layers: Process (P), Network (N), Control Logic or Runtime (C), Firmware (F), and Hardware (H).

1.3.1 Process Layer

Process layer defines the abstract model of the whole industrial process [8] as depicted in Figure 1.3. There are two commonly used models to describe physical processes: Auto-Regressive (AR) models, and Linear Dynamical State-space (LDS) models [137].

An AR model can be described as follows:

$$\hat{y}_{k+1} = \sum_{i=k-N}^{k} \alpha_i y_i + \alpha_0 \tag{1.1}$$

where coefficients α_i can be obtained through system identification and y_i represents the sensor measurement of the system at time i.

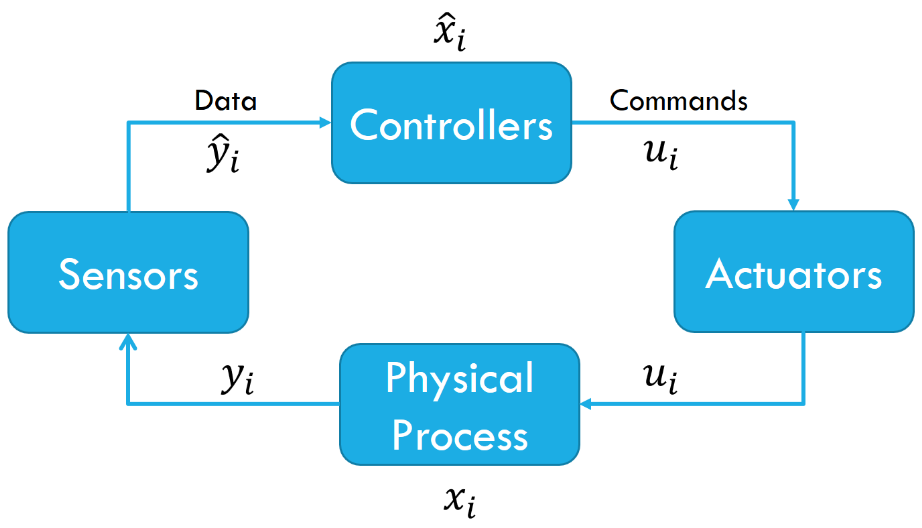


FIGURE 1.3: The flow chart of a process in a cyber-physical system.

An LDS model can be used when the system inputs (control commands) and outputs (sensor readings) are both available to the controller. Thus, we can use the following equations to model the dynamics of a linear system.

$$\hat{x}_{k+1} = Ax_k + Bu_k + \epsilon_k \tag{1.2}$$

$$\hat{y}_k = Cx_k + e_k \tag{1.3}$$

where x_i and y_i represent the state of the system and the sensor reading at time i respectively, A, B, C are three coefficient matrices that are obtained through system identification, u_i is the control input to the system at time i, ϵ_i represents the noise generated due to the imperfection of the system identification, and e_i denotes the measurement noise at time i.

In a control system, its abstract model (α_i in the AR model, or A, B, and C in the LDS model) is usually characterized beforehand. Based on the sensor measurement

$\hat{y}$ and the estimated state $\hat{x}$, the controller is able to take proper control actions and keep the system in the desired state.

1.3.2 Network Layer

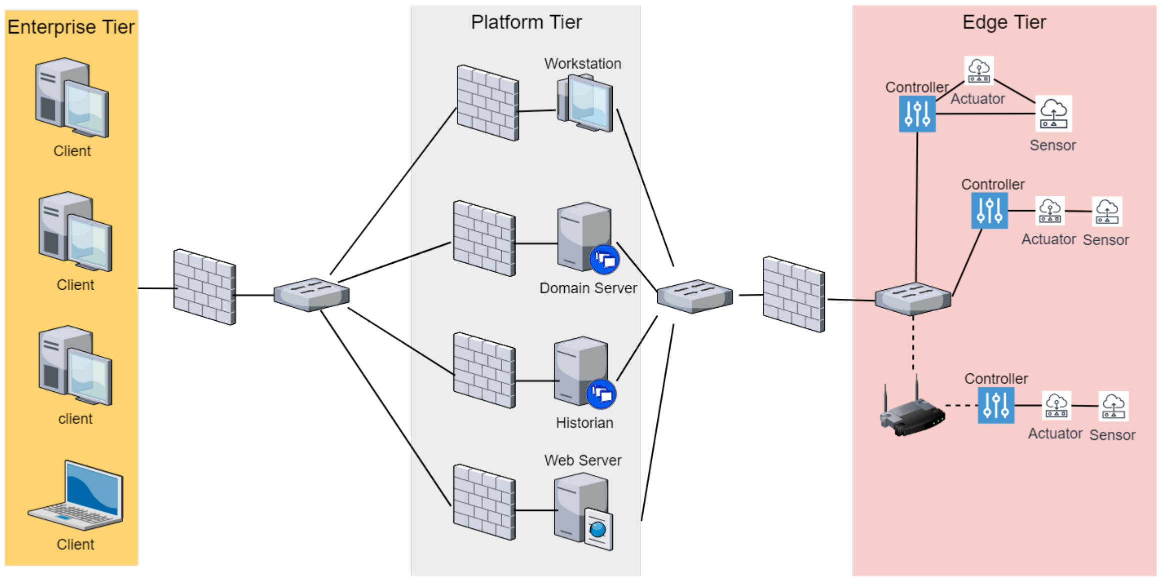


FIGURE 1.4: A typical network topology of industrial control systems.

Typically, an industrial control system, like a power grid, a water treatment system, or a manufacturing system, has a large number of devices connected together. All of them need to collaborate to correctly run the whole operation of the system. To this end, proper network infrastructure has to be in place, and a three-tier architecture depicted by Figure 1.4 is widely accepted [87, 175]. The architecture consists of edge tier, platform tier, and enterprise tier. Edge tier includes the devices which are at the edge of the system and directly control or interact with the physical processes, such as controllers, actuators, and sensors connected via local area networks. There are many ways to establish local area networks. The most common typologies are ring, bus topology, and wireless connections [183], as shown in Figure 1.4. The platform tier aggregates all the data from various edge devices, perform proper management and analysis on the data, and issue control commands to the devices in the field. The enterprise tier is the top tier in the architecture. It interacts with the platform tier, implements domain-specific applications and end-user interfaces.

1.3.3 Control Logic Layer

Control logic, sometimes also referred to as runtime or control program, is the program running on the programmable logic controllers (PLCs), which are the most commonly used controllers in an industrial control system [27]. Since PLCs and their control logic is designed for controlling physical processes, they have to frequently interact with the real world and react to the events captured by sensors very fast. This leads to a very different program execution model, comparing with the programs running on a conventional computer; the control logic on PLCs are running in an infinite loop, usually called a scan cycle. In each scan cycle, the PLC executes three steps sequentially: reading inputs from sensors, executing programs, and writing outputs to actuators. This execution model is depicted in Figure 1.5.

Since the control logic programming was designed to facilitate someone who does not have a great knowledge of programming, the International Electrotechnical Commission (IEC) defines the standard of the programming languages for PLCs. It includes five languages: Ladder diagram (LD), Function block diagram (FBD), Structured text (ST), Instruction list (IL) [1] and Sequential function chart (SFC) [86]. Among them, Ladder diagram and Structured text are widely used in practice. Lad-

[1]Instruction List was deprecated in the 3rd edition of the IEC 61131-3 standard [86].

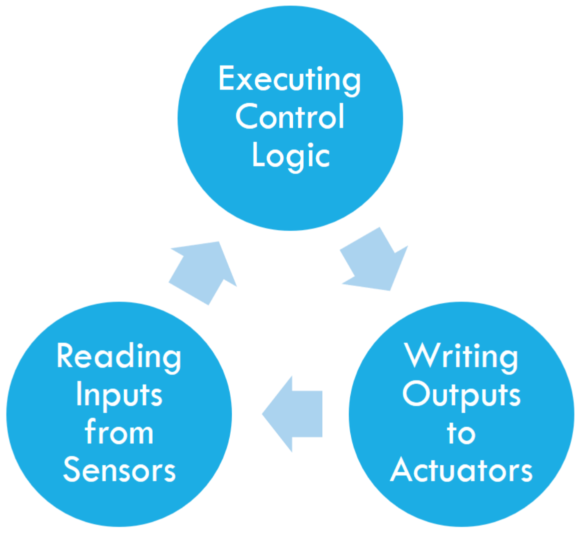


Figure 1.5: The execution model of a PLC.

der diagram is a graphical programming language, so it is widely used by the ones who do not have much experience in high-level language programming. An example is shown in Figure 1.6. In contrast, Structured text is very similar to high-level languages, as shown in Figure 1.7. Thus, it is widely adopted among who have programming experiences.

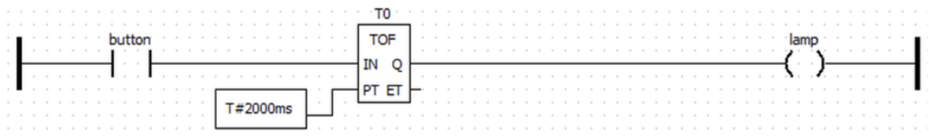
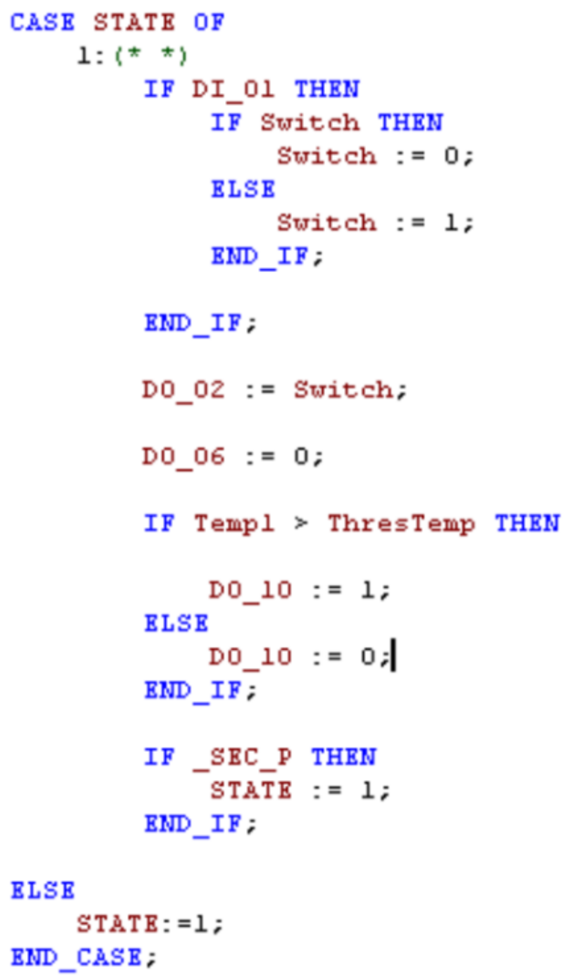


Figure 1.6: An example program using ladder diagram.

1.3.4 Firmware Layer

The firmware on the PLCs manages the hardware components on the PLCs. Particularly, it enforces the execution model of a PLC, such that all the inputs are read before one run of the control logic, and all the outputs are updated together after



```
CASE STATE OF
    1:(* *)
        IF DI_01 THEN
            IF Switch THEN
                Switch := 0;
            ELSE
                Switch := 1;
            END_IF;

        END_IF;

        DO_02 := Switch;

        DO_06 := 0;

        IF Temp1 > ThresTemp THEN

            DO_10 := 1;
        ELSE
            DO_10 := 0;
        END_IF;

        IF _SEC_P THEN
            STATE := 1;
        END_IF;

ELSE
    STATE:=1;
END_CASE;
```

FIGURE 1.7: An example program using structured text.

the control logic is completely executed. In a modern PLC, the firmware is also responsible for more tasks, including communication, operations of a web server, and redundancy management in case of a failure. Since there is a single core processor inside the PLC, these additional computational tasks will consume some computational power of the controller, and the PLC programmers should be aware of this extra performance overhead.

1.3.5 Hardware Layer

The hardware architecture in Figure 1.8 completely matches the execution model of the PLC control logic. It usually consists of three main components: input modules, a processor, and output modules. Additionally, Ethernet or wireless network modules are available in modern PLCs to enable collaboration with other PLCs and the management from a SCADA system.

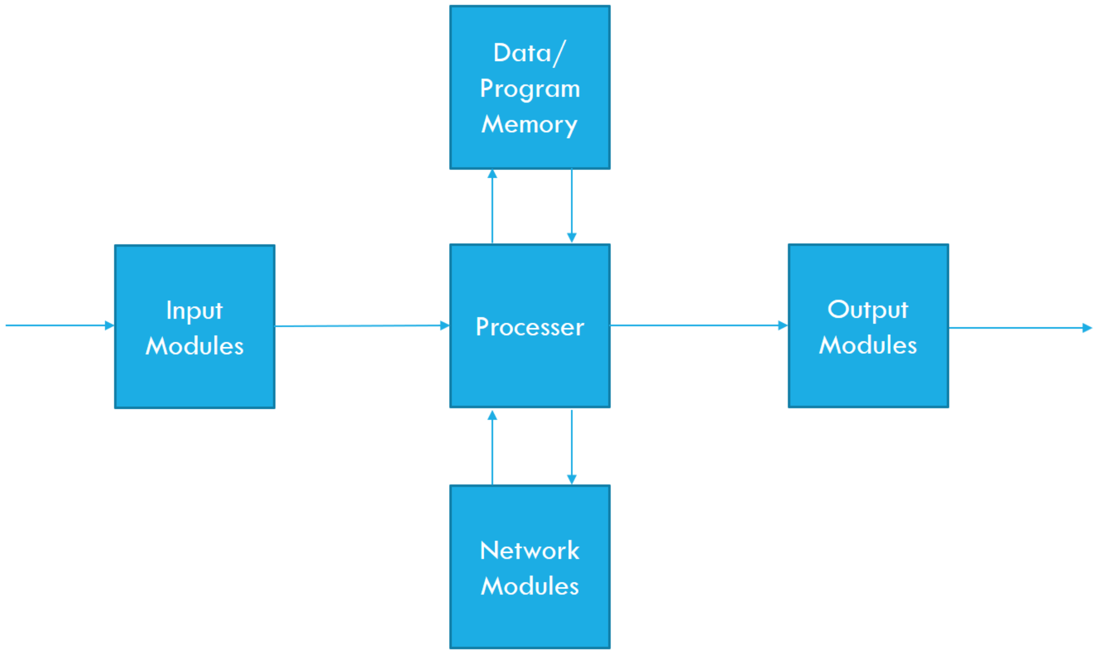


FIGURE 1.8: The hardware architecture of a PLC.

1.4 Adversarial Models

To build a framework which allows us to reason about security, we define adversarial models only according to the capabilities of an adversary in the system hierarchy, so adversaries are classified into five layers according to Figure 1.2. In each layer, we further characterize adversaries by two abilities: (1) the adversary is able to Fully compromise a particular layer or just Partially compromise it (denoted as F or P) ; (2) the adversary has Reading (R) or Writing (W) capability in the layer.

1.4.1 Process Layer Adversary $\mathcal{A}_P$

We call an attacker who has the capability of knowing all parameters of a running process an $\mathcal{A}_{P,F,R}$ attacker, where P, F, R denote "Process", "Full compromise", "Reading access", respectively. Similarly, $\mathcal{A}_{P,P,R}$ only knows some of the parameters. $\mathcal{A}_{P,F,W}$ denotes an adversary who can control every variable or equation of the process and essentially present a process that does not obey physical rules to the controllers, e.g., false data injection attack [136, 3]. In addition, $\mathcal{A}_{P,P,W}$ has limited capability in writing, so it can only compromise a certain fraction of all system variables like the limited access attack in [136].

1.4.2 Network Layer Adversary $\mathcal{A}_N$

Network layer adversary covers an extensive range of attack vectors, especially in the scenario of remote adversaries. A full compromise in the network layer means that the adversary can eavesdrop (reading access) or control (writing access) the network traffic in the whole network. However, in practice, certain security measures have been implemented in the industrial network. The most common security practices are firewalls and network segmentation [112, 113]. Firewalls can perform in-depth packet analysis and filter out a specific traffic patterns [129], and network segmentation can separate multiple sub-networks logically on the data link layer [112]. The firewalls and network segmentation (in the platform tier) are shown in Figure 1.4. These simple and efficient security measures in industrial networks constrain an adversary to be either $\mathcal{A}_{N,P,R}$ or $\mathcal{A}_{N,P,W}$ attackers who can only eavesdrop or tamper with the packets in a certain domain of the network. Note that, unlike the conventional network, generally speaking, a denial-of-service attack on network itself does not affect the operations of individual controllers and the physical processes. Thus network DoS attack is not an attack specific for cyber-physical systems, and therefore not included in our adversarial modeling.

1.4.3 Control Logic Layer Adversary $\mathcal{A}_C$

Since typically there are multiple devices/ controllers in an industrial control system, an attacker who can read/ write the control logic in all the controllers is considered as $\mathcal{A}_{C,F,R}$ and $\mathcal{A}_{C,F,W}$, respectively. In contrast, a partial compromise attacker is able to only compromise a fraction of all the controllers. Is it possible to partially compromise one controller's control logic, i.e., only a part of the control logic can be read by an adversary, and the other parts remain secrets to the adversary? To the best of our knowledge, there is no commercial PLCs which support this secret control logic feature. Also, note that when the control logic is compromised, usually the data (sensor inputs, actuator outputs) which is being processed by the control logic are exposed to the adversary as well. To maximize the uptime of a controller, commodity PLC vendors like Allen Bradley and Siemens usually allow users or adversaries to tamper with the control logic and the data when the controllers are running. In practice, many PLC attacks happened on the control logic layer [111, 73, 99, 24], including the most famous attack on industrial control systems, Stuxnet [65, 126, 122, 104].

1.4.4 Firmware Layer Adversary $\mathcal{A}_F$

The definition of a firmware layer adversary $\mathcal{A}_F$ is very similar to the control logic layer adversary, i.e., a partial compromise means the adversary can compromise a fraction of all controllers, while a full compromise adversary can take over the control of all of them. In practice, the firmware of the commodity PLCs are appropriately signed and encrypted, so it is not very easy for an attacker to get write access to this layer [73]. Although it is challenging to compromise the firmware of a PLC via network access, researchers have demonstrated that the firmware of an Allen Bradley PLC can be accessed, reverse engineered, and modified by having access to its JTAG port [70].

1.4.5 Hardware Layer Adversary $\mathcal{A}_H$

With the ongoing globalization of the electronics supply chain, more and more devices and chips are designed and fabricated by potentially untrusted parties [96]. Hardware Trojans are one of the most common ways to undermine the security of systems [80]. Although strong hardware Trojan detection tools, like HaTCh [79], do exist, it is still very costly to test all the chips and devices from an untrusted party. Hence, the security threats from the hardware layer to the whole cyber-physical system do exist [94, 135]. Similar to the adversarial modeling in the control logic layer and the firmware layer, the definition of partial/ full compromise still depends on the fraction of compromised devices. Writing capability means that a hardware Trojan is inserted or a legitimate device is replaced with a malicious one, and reading capability represents secret leakage from the device via side channel analysis [116, 115] or invasive probing attacks [81].

1.4.6 Cross Layer Adversary

Apparently, the above layer by layer classification is not sufficient to describe an attacker in the real world, as the real-world attacker can combine its capabilities in multiple layers to perform a more powerful attack. For the simplicity of our notation, we simply list the capabilities on all the layers where the attacker can compromise and separate them by a semicolon. For example, an adversary who can tamper with the control logic layer and firmware layer of all the devices can be denoted as $\mathcal{A}_{C,F,W;F,F,W}$.

1.5 Characterization of Real-World Attackers

There are a few types of real-world attack scenarios that are usually mentioned in some surveys of cyber-physical system security [32, 147, 1]. Below, we show how to characterize those commonly seen attacker models using our framework and therefore enable a more rigorous security analysis in the future.

1.5.1 Insider Attackers

Insider attackers are usually referred to as disgruntled or compromised employees who have a certain level of privileges in the system and want to perform attacks on the system [127]. Depending on their positions in the companies or facilities, insider attackers can have different attacking capabilities and fall into different categories of the above adversarial models.

For example, a malicious insider who only has authorization to access the control network but not the devices or computers physically [176, 119, 149] is an $\mathcal{A}_{N,F,W}$ or $\mathcal{A}_{N,P,W}$ adversary. Moreover, another insider attacker may be able to access the controllers physically but not the overall control system [123], which is categorized as an $\mathcal{A}_{C,P,W;F,P,W;H,P,W}$ adversary.

1.5.2 SCADA Attackers

Another real-world attacker model is so-called SCADA attackers, who can compromise the SCADA system completely and issues malicious commands and present fake "normal-looking" sensor data to operators. Most of the widely impacted cyber-physical system malware, such as Stuxnet [65, 145, 104], Industroyer [128], and Triton [99, 24, 169], fall into this category. For instance, in the case of Stuxnet, the malware first infected the workstations of the operators in Iranian nuclear plants, then the comprised workstation which runs SCADA system send malicious commands to the PLCs to alter the rotational speed of the centrifuges. Also, the operators are not aware of this incident because Stuxnet showed sensor data, which was recorded during normal operations on the screen [65].

Obviously, if an attacker compromises the SCADA system, then it has the exact same privilege as an operator, and thus, it can directly compromise the process, network, and control logic layers. This indicates that a SCADA attacker is an $\mathcal{A}_{P,F,W;N,F,W;C,F,W}$ adversary. If the attackers are sophisticated enough, they may even be able to compromise the firmware on controllers as well by exploiting some unpatched vulnerabilities of the PLCs [151, 12].

1.5.3 Single Stage Attackers

Due to the distributed nature of some industrial control systems, such as water distribution systems, gas distribution systems, and wind farms, every single stage or substation is not as well protected as the control center. In this scenario, a single stage attacker is more likely to occur. Usually, they can get access to a substation or a stage in the whole system, including all the controllers in the compromised stage [1, 178].

This kind of adversary can fall into many different categories depending on how the attacker tries to attack the whole system: (1) It can try to send false data to the controllers in other stages in the system [136]. This is an $\mathcal{A}_{P,P,W}$ adversary; (2) The attacker can also try to compromise other controllers and devices via the connected network [111]. This turns it into an $\mathcal{A}_{N,P,W}$ adversary; (3) Since the single stage attacker already has physical access to the whole stage, it can directly compromise anything on the devices [178, 1]. Thus, it is an $\mathcal{A}_{C,P,W;F,P,W;H,P,W}$ adversary.

1.6 Goals of Attackers

Besides the abstract adversarial models, it is also crucial for us to identify what is the purpose of an attack. In general, we classify all attacks, specifically targeting a cyber-physical system into two categories. The attacker either wants to compromise the confidentiality of the physical processes, or the integrity of the physical processes. Note that by limiting the goals on the physical processes, we explicitly exclude any attacks that can happen on any computation systems. For example, computers in a water plant in Pennsylvania were compromised by attackers in 2006, but they were just used to send spam emails [148]. This type of attack is not specific for cyber-physical systems, and therefore, it is not included in our classification.

1.6.1 Confidentiality of Physical Processes

Industrial control systems, or cyber-physical systems in general, vary from one to another. The components, topology, configurations, process parameters largely depend on the industry and the designer of individual systems. There are three purposes for an attacker to attack cyber-physical systems to compromise its confidentiality: attack preparation, industrial espionage, and user privacy threats. Notice that we discuss these three purposes separately, but in terms of adversarial modeling and countermeasure designs, these three purposes can be analyzed or prevented jointly because they all violate the confidentiality of the physical processes.

Attack Preparation. Due to the high diversity of industrial control systems, there is no generic attack which can be applied to any systems. One has first to understand how the system works before launching an attack to damage the system. Researchers have demonstrated how to exploit an Internet-facing PLC to build a network scanner to recover the whole network topology in the industrial control system [111]. PLC control logic binaries can be automatically reverse engineered by a malicious smartphone to facilitate future code injection attacks [107].

Industrial Espionage. Sometimes the industrial process itself is a trade secret to the company, especially in a manufacturing system [184]. How the components in the system work together and the values of the process parameters are supposed to be highly classified. However, attackers can successfully reconstruct the products that are being fabricated by smart manufacturing systems just by accessing acoustic side channel [66] or thermal side channel [5].

User Privacy Threats. In a smart metering system of a smart grid, the fine-grained energy consumption of every user is potentially threatened by malicious man-in-themiddle attackers or utility providers [146], and more personal data can be inferred from the leaked information [133]. Many privacy-preserving techniques have been proposed to solve this privacy issue [61, 164], including using differential privacy [56].

1.6.2 Integrity of Physical Processes

After the attacker has understood the system to a certain level, through either the above-mentioned methods or social engineering, an attack on the integrity of the physical processes can be launched. Similar to our discussion about the confidentiality, we identify three purposes for violating the integrity of processes, but a countermeasure that can protect the integrity of the system can probably address all the concerns below.

Denial-of-Service. A denial-of-service attack is frequently seen in the attacks on power grids, e.g., the cyber attack which caused a massive blackout in Ukraine in 2016 [128]. In such an incident, the attackers take over the control of the controllers and shut down all the services (open circuit breakers). More sophisticated attackers, like Industroyer malware, will wipe out the systems on the operators' computers, so it requires an even longer time to restore the service [128].

Physical Damage. Physical damage is a unique consequence of a cyber attack in cyber-physical systems. Stuxnet is one example of this kind of attacks [65, 145]. Stuxnet targets the centrifuges in Iranian nuclear plant, and it did quickly damage more than one thousand centrifuges in Iran just by operating them at an abnormal rotational speed [145]. Also, an attack can cause physical damage to the products produced by the processes, such that the products will have defects or inconsistent quality [184]. Last but not least, ransomware is another serious emerging security concern of cyber-physical systems. An attacker can threaten the company owners with physical damage to some parts in the system (e.g., turbines in a wind farm [178]) if they do not pay a ransom on time. Essentially, the root cause of such ransomware is that the attackers have the capability in creating physical damage to the system.

Human Safety Hazards. As another unique attack target in cyber-physical systems, an adversary who controls a CPS can introduce human safety hazards. A compromised industrial robot can potentially hurt the operators who work closely with it [162]. The safety instrumented system (SIS) of a petrol plant in Saudi Arabia got turned off by a cyber attack, which can cause an explosion in the plant [99, 169]. As cars are becoming smarter and smarter, their security loopholes can be exploited by attackers and lead to traffic accidents [54]. Besides, if a mass transportation system like railway systems got attacked, the consequences will be even more catastrophic [42].

1.7 Case Study of Past Real-World Incidents

In this section, we would like to present a case study of Stuxnet, which is the most famous attack on industrial control systems. Through this case study, we will show how a concrete adversary can be modeled in our framework and how to use our framework as guidance to find a solution and reason about its security.

1.7.1 Stuxnet

Background. Stuxnet is a computer worm which infects Windows computers and spread on its own to more targets [65]. It was designed to specifically target Siemens

PLCs running in nuclear plants in Iran, so its malicious behaviors will only be activated if it detects that the computer is connecting to a Siemens PLC which connects to at least 155 total frequency converters. This system configuration is unique to the nuclear plants in Iran, so it can be used as a signature to identify the targeted nuclear plant. Of course, this detailed intelligence requires significant reconnaissance efforts before the development of Stuxnet. No evidence shows how this information is leaked, but as we discussed earlier, it can be possible through cyber attacks or social engineering attacks.

It is widely believed that Stuxnet spread into the targeted nuclear plants in Iran through an infected flash drive. After it spread to the engineering workstation through the internal network inside the plant, its payload got activated, so it started measuring the rotational speed of the controlled centrifuges for 13 days and checked whether it is within the range from 800 Hz to 1200 Hz, which allows Stuxnet to make sure that it is attacking the correct target. Once the targets are confirmed by checking the rotational speed, Stuxnet downloaded malicious control logic to the Siemens PLCs. The designers of Stuxnet deliver the payload very carefully, so it can be unnoticeable by operators in the plant. The malicious control logic only sets the rotational speed to 1410 Hz for 15 minutes, and then it goes back to normal speed for 27 days. After that the rotational speed is set to extremely low at 2 Hz for 50 minutes, then again it runs at the normal speed for another 27 days. The whole process of altering the rotational speed is repeated forever. By running the centrifuges at an extreme speed, the centrifuges got damaged very fast, and a hardware replacement is needed. This process significantly slowed down the Iranian nuclear plan.

To further hide its malicious behavior from the operators in the control room, a recorded normal rotational speed measurement will be displayed on the screen of the

SCADA system when the speed is altered. In such a way, even the SCADA system itself cannot detect the violation of the integrity of the physical processes.

Formalization. Using our framework, we can formalize the security game of a Stuxnet adversary $\mathcal{A}_{P,F,W;N,P,W;C,F,W}$[2] who plays with a defender $\mathcal{D}$ which outputs 1 if it detects attacks (for simplicity, we will use $\mathcal{A}$ to represent the Stuxnet adversary thereafter):

```
1:  **Game** STUXNET($\mathcal{A}$, SPEC)
2:      A system $S$ with state $x_0 \leftarrow \mathsf{SPEC}$
3:      A defender $\mathcal{D} \leftarrow \mathsf{SPEC}$
4:      **for** every time unit $i$ **do**
5:          Given access to the SPEC, $\mathcal{A}$ generates a control logic $CL$, a network packet
        $NP$, new sensor readings $y'_i$, and the information ($I_{\mathcal{D}}$) which will be sent to $\mathcal{D}$.
6:          $(u_{i+1}, x_{i+1}) \leftarrow S(CL, NP, y'_i, x_i)$
7:          **if** $(x_{i+1} \notin \mathsf{SPEC}_{i+1}$ or $u_{i+1} \notin \mathsf{SPEC}_{i+1})$ and $\mathcal{D}(I_{\mathcal{D}}) = 1$ **then**    ▷ $x_{i+1} \notin$
        $\mathsf{SPEC}_{i+1}$ means $x_{i+1}$ does not follow SPEC at $i+1$
8:              halt $S$ and break
9:          **end if**
10:     **end for**
11:     Return $i$
12: **Game end**
```

The attacker's goal is to maximize i, while the defender would prefer a small i. Also, as the system S is a physical system which is subject to random noise, a given pair of $\mathcal{A}$ and $\mathcal{D}$ will result in a probability distribution of i, and an output i is drawn

[2] $\mathcal{A}$ has partial writing capability on the network because the adversary can only control the network packets having sources or destinations as the compromised machine.

from the distribution after every run of the game.

Summary of the Case Study. From the security game we presented above, it is clear to see that the information which will be sent to the detector D is very crucial to the effectiveness of the system. I should only contain information that cannot be tampered and controlled by the adversary, i.e., the network traffic not sent to or from the SCADA system, the firmware information, and the hardware information. In other words, it means that to prevent such an adversary, we have to rely on the information which is outside of the control of the adversaries. This also shows that through such adversarial modeling and formalization, we can easily find the direction of the solution and reason about its security. Of course, a more fine-grained adversarial modeling allows one to identify more information that is beyond the control of the adversaries, which can improve the existing solutions.

1.8 Outline of the Dissertation

This dissertation introduces several cryptographic solutions for the security issues in cyber-physical systems.

1. We designed and implemented a sensor fusion system that can formally defend against pollution attacks without compromising the privacy of individual sensor data in Chapter 2 [97]. According to our framework, the adversary is $\mathcal{A}_{P,P,W;N,F,R}$ adversary who wants to violate the confidentiality and integrity of the process altogether.

2. In Chapter 3, a lightweight intrusion detection system for industrial control systems is introduced [95]. It leverages forward secure key management scheme

to secure the log of the programmable logic controllers in the presence of the strongest adversarial model. Its adversarial model is $\mathcal{A}_{N,F,W;C,F,W}$. From the adversarial modeling, we can see that our solution has to be implemented on the firmware or hardware layer.

3. An authenticated key exchange protocol is introduced in Chapter 4 [98]. It utilizes historical data of the industrial process as the second authentication factor to enhance the security of a secure channel establishment between a PLC and the SCADA server. As an authenticated key exchange protocol, its typical adversarial model is $\mathcal{A}_{N,F,W}$ in our framework.

1.9 Publications

Journal papers that are accepted and published with primary authorship include [96, 93]:

1. **C. Jin** and M. van Dijk, "Secure and Efficient Initialization and Authentication Protocols for SHIELD". *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 1, pp. 156-173, 2019.

2. **C. Jin**, C. Herder, L. Ren, P. H. Nguyen, B. Fuller, S. Devadas, and M. van Dijk, "FPGA Implementation of a Cryptographically-Secure PUF based on Learning Parity with Noise." *Cryptography*, vol. 1, no. 3, pp. 23, 2017.

Currently submitted journal papers with primary authorship include [98]:

1. **C. Jin**, Z. Yang[3], S. Adepu, and J. Zhou, "HMAKE: Legacy-Compliant Multi-

[3]C. Jin and Z. Yang share the first authorship and are ordered alphabetically.

factor Authenticated Key Exchange from Historical Data," *IEEE Transactions on Dependable and Secure Computing.* 2019.

Conference papers that are accepted and published with primary authorship include [95, 94]:

1. **C. Jin**, S. Valizadeh, and M. van Dijk, "Snapshotter: Lightweight Intrusion Detection and Prevention System for Industrial Control Systems," in *IEEE Industrial Cyber-Physical Systems*, 2018, pp. 824-829.

2. **C. Jin**, L. Ren, X. Liu, P. Zhang, and M. van Dijk, "Mitigating Synchronized Hardware Trojan Attacks in Smart Grids," in *Proceedings of the 2nd Workshop on Cyber-Physical Security and Resilience in Smart Grids.* ACM, 2017, pp. 35-40.

Journal papers that are accepted and published with co-authorship include [156, 79, 75, 78]:

1. P. H. Nguyen, D. P. Sahoo, **C. Jin**, K. Mahmood, U. Rührmair, and M. van Dijk, "The Interpose PUF: Secure PUF Design against State-of-the-art Machine Learning Attacks". In *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019.

2. S. K. Haider, **C. Jin**, M. Ahmad, D. Shila, O. Khan, and M. van Dijk, "Advancing the state-of-the-art in Hardware Trojans Detection," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 1, pp. 18-32, 2019.

3. X. Guo, **C. Jin**, C. Zhang, A. Papadimitriou, D. Hély, and R. Karri, "Can Algorithm Diversity in Stream Cipher Implementation Thwart (Natural and)

Malicious Faults?", *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 3, pp. 363-373, 2016

4. X. Guo, D. Mukhopadhyay, **C. Jin**, and R. Karri, "Security Analysis of Concurrent Error Detection against Differential Fault Analysis," *Journal of Cryptographic Engineering*, vol. 5, no. 3, pp. 153-169, 2015.

Conference papers that are accepted and published with co-authorship include [109, 185, 191, 80, 139, 76, 77]:

1. R. S. Khan, N. Noor, **C. Jin**, S. Muneer, F. Dirisaglik, A. Cywar, P. H. Nguyen, M. van Dijk, A. Gokirmak, and H. Silva. "Exploiting Lithography Limits for Hardware Security Applications.", in *2019 IEEE Conference on Nanotechnology*, 2019.

2. M. van Dijk, **C. Jin**, H. Maleki, P. H. Nguyen, and R. Rahaeimehr, "Weak-Unforgeable Tags for Secure Supply Chain Management," in *2018 International Conference on Financial Cryptography and Data Security*, 2018

3. W. Yan, **C. Jin**, F. Tehranipoor, and J. A. Chandy, "Phase Calibrated Ring Oscillator PUF Design and Implementation on FPGAs," in *Field Programmable Logic and Applications, 2017 27th International Conference on.* IEEE, 2017, pp. 1–8.

4. S. K. Haider, **C. Jin**, and M. van Dijk, "Advancing the state-of-the-art in Hardware Trojans Design," in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems.* IEEE, 2017, pp. 823-826.

5. H. Maleki, R. Rahaeimehr, **C. Jin**, and M. van Dijk, "New Clone-Detection Approach for RFID-based Supply Chains," in *Hardware Oriented Security and Trust, 2017 IEEE International Symposium on.* IEEE, 2017, pp. 122-127.

6. X. Guo, N. Karimi, F. Regazzoni, **C. Jin**, and R. Karri, "Simulation and Analysis of Negative-Bias Temperature Instability Aging on Power Analysis Attacks," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust.* IEEE, 2015, pp. 124-129.

7. X. Guo, D. Mukhopadhyay, **C. Jin**, and R. Karri, "NREPO: Normal Basis Recomputing with Permuted Operands," in *Hardware-Oriented Security and Trust, 2014 IEEE International Symposium on.* IEEE, 2014, pp. 118-123

Book chapters that are accepted and published with co-authorship include [110, 182]:

1. R. S. Khan, N. Noor, **C. Jin**, J. Scoggin, Z. Woods, S. Muneer, A. Ciardullo, P. H. Nguyen, A. Gokirmak, M. van Dijk, and H. Silva, "Phase Change Memory and Its Applications in Hardware Security," in *Security Opportunities in Nano Devices and Emerging Technologies.* CRC Press, 2017, pp. 115-136.

Chapter 2

PwoP: Intrusion-Tolerant and Privacy-Preserving Sensor Fusion

2.1 Introduction

Numerous modern cyber-physical systems (CPS), spanning industry, agriculture, military, and beyond, are increasingly relying on distributed data sources (hereinafter without loss of generality, sensors) to support critical decisions and actions. As depicted in Figure. 2.1, the integration of these sensor data are most often achieved with the help of a server (proxy, aggregator, or averager), which upon receiving a client request, gets sensor inputs from a set of sensors, integrates the sensor inputs, and returns the result to the client. The client may then inform the actuator what to do. Its application scenarios are almost everywhere, including sensor networks, smart metering, GPS devices and satellites, soldiers in battlefields, smart phones and the cloud, time-keeping mechanisms [144], and so on.

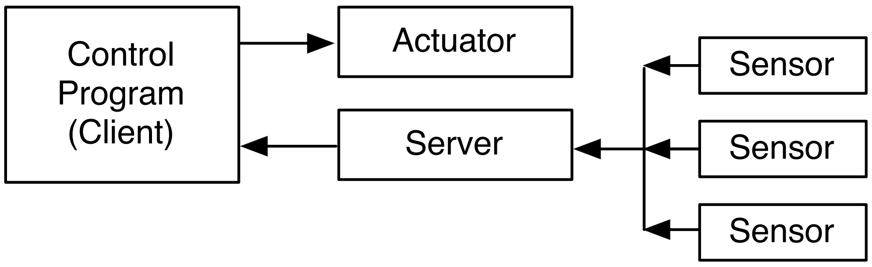


FIGURE 2.1: Distributed sensor fusion architecture.

Privacy and integrity are widely regarded as primary concerns or even hurdles for many of these applications [186]. First, we need to protect the privacy of individual data sources. Ideally, clients should learn only the designated function of the sensor inputs but nothing more, and the server should learn nothing (about sensor inputs or the final result). Second, we require integrity, in the sense that the server should faithfully return to the client the correct, integrated result.

Pollution attacks. An equally important but notoriously difficult goal in multi-sensor fusion is to defend against *pollution attacks*, where some malicious sensors lie about their values to sway the final result. Specifically, motivated attackers can mount this kind of attack by either corrupting sensors and contributing malicious inputs, or maliciously altering environmental variables (say, by manipulating the environmental values for sensors without actually corrupting the sensors themselves).

At first glance, defending against pollution attacks seems to be at odds with attaining privacy. Indeed, to achieve the strongest privacy goal mentioned above, the server is not supposed to learn individual sensor inputs. Thus, in spite of the risk of pollution attacks, the vast majority of existing privacy-preserving systems treat defending against pollution attacks as out-of-scope.

Yet still, there are a handful of prior works attempting to *mitigate* the problem [124, 47]. Their approach is to ask the sensors to provide a cryptographic proof to show their inputs are in a prescribed range (or more generally satisfying some predicate), in the hope that a coalition of malicious sensors would not affect the final result by much. Take the average function for example. Suppose we have ten sensors, each of which can have an input selected from the range [1, 100]. Also assume that the "correct" value is around 20 and correct sensors will output a value around this. If three malicious sensors contribute 100 (which is in the range), they would introduce a significant error into the final result. Moreover, for many applications, there are no prescribed limits on sensor inputs. To the best of our knowledge, all existing privacy-preserving aggregation and fusion schemes are vulnerable to pollution attacks to a significant degree.

Working in computation and bandwidth restricted CPS. Despite an impressive amount of work on secure sensor fusion or data aggregation protocols [40, 100, 36, 91, 121, 62, 172, 199, 47, 59, 150], to the best of our knowledge, none of them are implemented in real computation and bandwidth-restricted environments. They either do not provide any implementation, or their systems are run using commodity computers or virtual machines. These systems do not consider computation and bandwidth restricted environments specifically, and a new design with these factors considered is needed. Meanwhile, implementing a real system in these restricted environments involves integration of knowledge and expertise from different areas — hardware, software, and network communications.

Our approach. We design and implement, PwoP ("Privacy w/o Pollutions"), a privacy-preserving and fault-tolerant sensor fusion system that 1) defends against pollution attacks, 2) performs within the computation and bandwidth limitations of cyber-physical systems, 3) covers different application scenarios and different sensor input types, and 4) is efficient and scalable in both failure-free and failure scenarios.

To defend against pollution attacks, instead of relying on validity proofs, our strategy is to "tolerate," so that no matter what inputs malicious sensors provide, the fused value represents the correct physical value with good accuracy, still in a privacy-preserving manner.

Specifically, PwoP combines techniques from distributed systems and multi-party computation. At the core of PwoP are a series of practical sensor fusion algorithms that tolerate malicious inputs themselves [142, 144, 167, 45, 30]. For example, one of Marzullo's algorithms [142] ensures that the range produced by fusing ranges provided from sensors contains the actual value measured by the sensors if at most f out of $2f + 1$ sensors are malicious. We extend the framework of Yao's garbled circuits (GC) for two-party computation to the client-server-sensors setting for these fault-tolerant algorithms, leveraging their built-in fault-tolerance to efficiently achieve system liveness (i.e., guaranteed output delivery).

From the system perspective, we build a new and *general* compiler enabling the multi-sensor setting from TinyGarble [177]. We also make significant tailored optimizations, yielding a system that performs considerably better than direct application of GC compilers produce. Our system also tackles various other security, availability, and reliability concerns.

PwoP covers a variety of sensor input types, including intervals, rectangles, and d-dimensional inputs, and sensors inputs with bounded and unbounded accuracy. Moreover, our system works well in low bandwidth environments, thereby capturing a wide range of real sensor applications.

System assumptions. Throughout the paper, we rely on two assumptions:

- To achieve meaningful robustness against pollution attacks, the number of malicious sensors should be bounded (less than 1/2 or 1/3 of the total number, depending on concrete applications and algorithms).
- While both the server and a fraction of sensors may be malicious, the server should not collude with malicious sensors [101, 103]. (We stress that in our system sensors can collude, and in fact, we consider a strong adversary can coordinate malicious sensors to compromise the system.)

The first assumption is a standard one in distributed systems and multi-party computation systems where sensors, ideally, are independently distributed in different hosts (running diverse software and hardware), and the adversary has only limited capacity to compromise the system overall.

The second assumption is also a common assumption that has been used in a large number of practical multi-party computation systems [101, 103, 34, 33, 153, 199, 47]. Our system therefore is suitable for applications where the server and sensors lack motivation to collude, or the adversary lacks the means to corrupt both the server and some sensors simultaneously. For example, a satellite does not own the GPS devices, and the adversary may lack means to compromise the satellite and some GPS devices together. Another example is that soldiers with sensors may be captured by the enemies who may be unable to compromise the headquarter, which is typically well protected. As another one, the assumption makes sense for submarines and underwater sensors.

Our contributions. We summarize our contribution as follows:

- We motivate, clarify, and formalize the problem of server-aided, privacy-preserving, and intrusion-tolerant multi-sensor fusion.

- We provide an efficient and expressive framework supporting a variety of key fault-tolerant sensor fusion algorithms. The framework opportunistically leverages the bandwidth and computation asymmetry property in the sensor fusion setting and uses a novel combination of techniques from distributed systems (Byzantine fault-tolerant sensor fusion [142]) and multi-party computation (Yao's garbled circuits [196]). In addition, we use new techniques for achieving system liveness (which would otherwise leverage slower solutions using more than one server), make extensive optimizations on the framework, and tackle various other security and reliability issues.
- We make a general compiler specifically for our clients-server-sensors setting by extending and optimizing TinyGarble [177]. TinyGarble is designed for two-party computation (for garbler and evaluator), but our compiler works for settings involving clients (garblers), server (evaluator), and sensors (garbled input providers).
- We build a practical system for server-aided multi-sensor fusion using Raspberry Pi Zero W. Our experimental evaluation demonstrates that the system is highly efficient and scalable for *both* failure-free *and* failure scenarios.

2.2 Related Work

Comparison with differentially-private systems. Apple [13] and Google [63] use differential privacy [56, 57] mechanisms to compute aggregate statistics. In these systems, each sensor adds random noise and the noisy data will be aggregated to get an estimate of aggregated values. These systems allow us to achieve robustness, but have to trade between accuracy and privacy. Increasing the noise level reduces information leakage for individuals, but also reduces the estimated accuracy.

Essentially, the goals between differentially-private systems and our system are orthogonal. In a differentially-private system, the server will know a noisy version of the value measured by each sensor, and thus learn much more than that in our system. However, the client learns slightly less than our system because of the noise added.

Comparison with privacy-preserving data fusion and aggregation. Most privacy-preserving data aggregation systems only protect individual sensor inputs but fail to handle malicious sensors that attempt to sway the aggregated value [40, 100, 36, 91, 121, 62]. Only a handful provide a partial solution by leveraging a cryptographic proof that sensor input has a specific property (say, is within a predetermined range) [172, 199, 47]. PwoP takes a fundamentally different approach by *tolerating* malicious sensor inputs without asking for input validity proofs. Other works (e.g., [40, 91]) provide privacy-preserving aggregation with tolerance to benign (crash) sensor failures, i.e., where some sensors fail to provide their inputs. Instead, our system deals with Byzantine failures, where corrupted sensors can provide the server arbitrary values.

A number of privacy systems [59, 150, 172, 91] additionally provide differential privacy. These systems still do not formally defend against pollution attacks.

SIA [160] explored a setting where individual sensor inputs do not need to be privacy-protected but the central server needs to *verifiably* provide clients with correct values. This helps achieve integrity. In contrast, PwoP achieves both privacy and integrity.

Comparison with alternative approaches to intrusion-tolerant and privacy-preserving sensor fusion. In addition to GC-based approach, we also provide alternative solutions with the same goal as PwoP: one using order-preserving encryption, and one using set representation. We describe the two approaches in Appendix 2.9, and compare them with PwoP in detail. Summarizing, the set-representation-based approach only applies to honest-but-curious sensors and a very limited number of fusion algorithms, and the approach based on order-preserving encryption leaks too much information and cannot easily achieve integrity.

Fault-tolerance and garbled circuits. Nielsen and Orlandi [157] built LEGO for two-party computation in the malicious case. In LEGO, the garbler first sends many gates, and the receiver tests if they are constructed correctly by opening some of them. Then the parties run interactively to solder the gates (as Lego blocks) into a circuit. They use a fault-tolerant circuit to ensure a valid output from a majority of good ones. In contrast, our system exploits the fault tolerant features of the underlying algorithms to achieve a garbled circuit based, privacy-preserving system that can tolerate pollution attacks, returning correct results even in the presence of Byzantine failures and malicious attacks.

Non-colluding multi-party computation. The assumption that a number of parties do not collude is not only used to build theoretical multi-party computation [16, 23, 72, 101], but used in practical multi-party computation systems [47, 199, 103, 34, 33, 153]. Among these systems, many use garbled circuits as a building block (e.g., [101, 103, 34, 33, 153]). Our notion of non-collusion follows this line of research, but has an architecture that is different from all these existing systems. In our setting, we assume that clients only have means to contact the server, and we assume the server and sensors do not collude. Note that non-collusion of parties does not imply that the parties are trusted. Rather, it simply means these parties do not work together (i.e., share internal states).

A few systems [33, 153, 34] relying on the non-colluding assumption work in the three-party computation only, using modern mobile devices. They attempt to resolve different problems from ours.

Efficient GC implementations. Starting from Fairplay [141], a large number of GC tools (compilers or implementations) have been proposed [134, 177, 85, 18, 200, 120]. Our system is based on (but makes significant modifications to) TinyGarble [177], an approach that in addition to using state-of-the-art optimizations such as free-XOR [118], row reduction [154], fixed-key AES garbling [18], and half gates [201], leverages logic synthesis to reduce the size of circuits.

Related attacks. Pollution attacks have also been studied in other areas such as network coding [2] and personalized services [190]. We work with a fundamentally different setting, focusing on data and sensor pollution attacks.

Our pollution attack scenarios are also different from those of Sybil attacks [52] where an adversary may forge multiple or even an unlimited number of identities to damage distributed systems. While Sybil attacks and pollution attacks may share somewhat the same goal, Sybil defenses [11] offer no help for defending against data pollution.

2.3 System and Threat Model

The setting. As depicted in Figure. 2.1, our system, PwoP, consists of a number of clients (control programs), a single server (proxy, averager, aggregator), and a set of sensors. Client and server are denoted c and S respectively. We denote the number of sensors by n, and a bound on the number of faulty sensors by g. The set of sensors is denoted as $\Pi = (s_1, \cdots, s_n)$. Let l be the length of the sensor input. Let k be the security parameter of cryptographic primitives.

In PwoP, a client sends a request to the single server, and the server collects readings from some or all of the sensors. Then the server runs a sensor fusion algorithm and sends the aggregated result to the client. Clients and sensors only communicate with the single server. In particular, sensors neither need to know each other, nor send one another any information.

Throughout the paper, we assume authenticated and private channels.

Threat model. A participant (client, the server, or sensor) that faithfully executes our protocol to completion is said to be *correct*; otherwise it is *Byzantine* or *malicious*. The behaviors of malicious participants are limited only by the cryptographic assumptions that we employ. A Byzantine participant that conforms to the protocol until some point at which it simply stops executing (permanently) is said to *crash*. A correct participant can nevertheless be *semi-honest*, namely it conforms to the protocol but may additionally preserve the transcript of everything it observes, in an effort to glean information to which it is not entitled. Participants (Byzantine or semi-honest) may also be required to be *non-colluding* (e.g., [101, 103]), which informally means that they do not share information except as explicitly prescribed by the protocol. The non-colluding assumption has been used in many practical multi-party computation systems [103, 34, 33, 153].

In PwoP, clients are semi-honest, while the server and some sensors can be malicious. We have designed but have not implemented an enhanced system that can defend against malicious clients; see Sec. 2.9.

Goals. PwoP aims to achieve privacy, correctness, and liveness.

- **Privacy**: If a semi-honest client does not collude with the server and does not collude with sensors, then it learns only the aggregated result returned from the server but nothing more. If the server does not collude with the client and does not collude with sensors, then it learns nothing about sensor inputs or the final result.
- **Correctness**: If the server is correct and no more than g sensors are Byzantine, then the only response a correct client will accept is the correctly aggregated result, in the sense discussed in the next section.
- **Liveness**: If the server is correct, if no more than g sensors are Byzantine, and if all Byzantine sensors crash, then each correct client receives a reply to its request.

2.4 Fault-Tolerant Sensor Averaging Algorithms

We briefly survey the key fault-tolerant sensor fusion/averaging algorithms [143, 144, 142, 167, 45].

Marzullo's algorithms [142, 144]: M-g-U, M-g, M-g-m, and M-op. The study of fault-tolerant sensor averaging algorithms dates to two seminal works by Marzullo [142, 144]. We collectively call them Marzullo's algorithms.

In the presence of n sensor values from n replicated sensors, a *fault-tolerant sensor averaging algorithm* [142] is used to compute a correct aggregated value even if some of the individual sensors are incorrect (in which case the sensor is said to be malicious, Byzantine, or simply faulty). Marzullo [142] considered the case where each individual sensor value can be represented by an interval $I = [u, v]$ over the reals. Let $(u - v)$, the width of the interval, denote the *accuracy* (or *inaccuracy*) of a sensor. Let $(u + v)/2$ be the *midpoint* or *center* of the interval. Then, a sensor value is correct if the interval I it returns contains the actual value of the measured feature, and the sensor is faulty otherwise. The goal of Marzullo's algorithm is to find the minimum (and correct) interval given n different intervals $I = \{I_1, \cdots, I_n\}$, with at most $g < n$ of those being faulty. The fused interval is at least as accurate as the range of the *least* accurate individual non-faulty sensors.

We start by introducing algorithms where the number of failed sensors g is known. The underlying idea is follows: Since g or less sensors are incorrect, any $(n - g)$ mutually intersecting sensors (i.e., *clique*) may contain the correct value. The algorithm computes the "cover" (not the "union") of *all* $(n - g)$-cliques.[1] Let lo be the smallest value contained in at least $n - g$ of the intervals and hi be the largest value contained in at least $n - g$ of the intervals. Then, the correct aggregated result is the interval $[\mathsf{lo}, \mathsf{hi}]$.

Marzullo [142] describes a general algorithm with $O(n \log n)$ complexity to compute this result. It uses a *sweeping* idea: First, sort all the endpoints of all the intervals. Second, moving from the lowest value to the highest value, keep track of the number of intervals containing each value. The final result can then be determined from these counts. The algorithm cost is dominated by the sorting procedure. Additional care is needed when the rightmost endpoint value of one interval coincides with the leftmost one of another interval, indicating that one interval ends exactly as another begins. Whether such an occurrence is deemed as a valid duration may

[1]Picking the cover instead of the union can help preserve the shape of the sensor value.

depend on applications, but this should be agreed upon beforehand. In this paper we follow Marzullo's convention [142], which considered the occurrence as being valid.

The above algorithm can apply to the case of arbitrary failures with unbounded inaccuracy, and to the case of arbitrary failures with bounded inaccuracy, where the maximum length of the interval is known and values that are too inaccurate can be detected. Marzullo's algorithm needs $3g+1$ and $2g+1$ sensors to tolerate g arbitrary failures with unbounded inaccuracies and bounded inaccuracies, respectively. We used M-g-U and M-g to denote the above two cases.

It is not uncommon to require the averaging algorithm to only return the midpoint of the interval. This may even be more desirable in a privacy-preserving setting, as providing the lo and hi values might reveal too much unnecessary information. We write M-g-m to denote this variant.

Marzullo [144] also gave a solution to the case where the system parameter g is unknown or unspecified. The goal is to find the cover for the *maximum* intersection groups for all the intervals. Thus, the algorithm is "optimistic" (instead of "optimal"), and we write M-op to denote this one.

Schmid and Schossmaier [167]: SS. Marzullo's algorithms may exhibit a somewhat irregular behavior: it is possible that when Marzullo's algorithms are applied to two slightly different input sets, the output may be quite different. This is formalized as violation of the *Lipschitz condition* regarding a certain metric [125].

Schmid and Schossmaier [167] offered a solution, SS, which can satisfy the Lipschitz condition. The algorithm is simple: Given n intervals $I_i = [u_i, v_i]$ $(1 \leq i \leq n)$, (at most) g of which may be faulty, SS simply outputs $[\max^{g+1}\{u_1, \cdots, u_n\}, \min^{g+1}\{v_1, \cdots, v_n\}]$, where $\max^{g+1}\{u_1, \cdots, u_n\}$ denotes the element $u_{j_{g+1}}$ in the or-

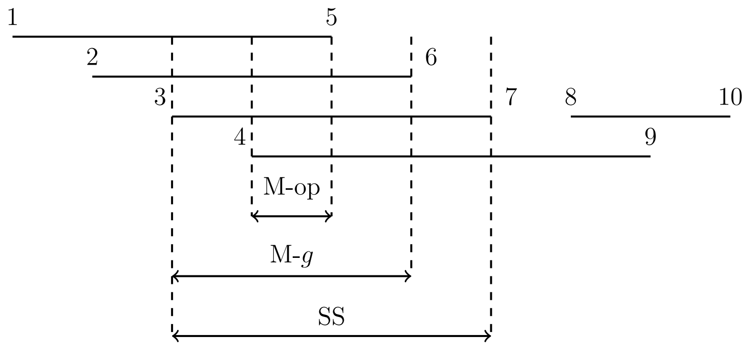


FIGURE 2.2: A five-sensor system with M-g, M-op and SS. The sensor input intervals are [1, 5], [2, 6], [3, 7], [4, 9], [8, 10]. The resulting intervals are [3, 6], [4, 5] and [3, 7] for M-g, M-op and SS respectively.

dering $u_{j_1}, \cdots, u_{j_n}$ of $\{u_1, \cdots, u_n\}$ from largest to smallest, and $\min^{g+1}\{v_1, \cdots, v_n\}$ denotes the element $v_{j_{g+1}}$ in the ordering $v_{j_1}, \cdots, v_{j_n}$ of $\{v_1, \cdots, v_n\}$ from smallest to largest. While SS shares the same worst case performance as Marzullo's, SS may generate a larger output interval.

Chew and Marzullo [45]: ChM. Chew and Marzullo generalized the approach to handle the case of *multidimensional* values. We will cover the most efficient algorithms described in their paper, including an algorithm for general d-dimensional rectangles (ChM-dD) and an algorithm for general d-dimensional rectangles that have the same size and orientation (ChM-dD-sso). To tolerate g failed sensors, these algorithms use $2dg + 1$ and $2g + 1$ sensors respectively. Note that for the case of ChM-dD-sso, the total number of sensors n does not depend on the dimension d. We defer the concrete description of these algorithms to where we need them.

Example. To help understand the algorithms described, we describe an example in Figure. 2.2 which shows how three one-dimensional algorithms (M-g, M-op, and SS) work. All the three algorithms deal with bounded accuracy and use $2g+1$ sensors to tolerate g failures.

As in Figure. 2.2, the input intervals for the sensors are [1, 5], [2, 6], [3, 7], [4, 9] and [8, 10]. For all the algorithms, all input endpoints need to be sorted.

To find the left endpoint of the resulting interval for M-g, we can imagine that there is a vertical line sweeping from left to right. The vertical line can stop at the leftmost point that intersects $n - g = 3$ intervals. In the example, this point is 3. Similarly, to find the right endpoint, a vertical line can sweeping from right to left, and find the right end of the resulting interval (6). Thus, the resulting interval is [3, 6].

Instead of outputting an interval, M-g-m will output the midpoint of the resulting interval generated by M-g.

In contrast to M-g, M-op algorithm does not need to know the g value a-priori. A vertical line will sweep over all the endpoints and finds the leftmost and rightmost points that intersect with the maximum input intervals. In the example, point 4 and 5 are covered by four input intervals, while the rest endpoints are covered by at most three input intervals. Thus, M-op will output [4, 5] as the result.

For SS, one need to find the $(n - g)$-th smallest left endpoint and the $(n - g)$-th largest right endpoint. In the example, point 4 and point 7 are picked as they are the third smallest left end and the third largest right end, respectively.

The example would be easily extended to explain M-g-U with unbounded accuracy. However, it requires at least $3g + 1$ sensors to tolerate g failures.

In Appendix A.3, we show an example on how d-dimensional algorithms (ChM) work.

2.5 PwoP

This section presents PwoP. We first describe the key building block of PwoP—garbling schemes [19]—and then the design of PwoP.

2.5.1 Garbling Schemes

Bellare, Hoang, and Rogaway (BHR) [19] introduced the notion of a *garbling scheme* as a first-class cryptographic primitive. Here we mainly adopt this abstraction but tailor it for our purposes; specifically, we require that *all* the garbling scheme algorithms be *dominated* by random coins. The change is only notational.[2]

A garbling scheme is a tuple of algorithms $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$. Gb takes as input 1^k, a random coin r and a Boolean circuit f, and outputs a garbled circuit F. En takes an input x and a random coin r and outputs a garbled input X. Ev takes a garbled circuit F and garbled input X and outputs a garbled output Y. De takes a garbled output Y and a coin r and outputs a plain-circuit output y (or $\perp$).

We require a correctness condition on garbling schemes: if $\mathsf{F} \leftarrow \mathsf{Gb}(1^k, r, f)$, then $\mathsf{De}(r, \mathsf{Ev}(\mathsf{F}, \mathsf{En}(r, x))) = f(x)$.

In our work, we require the prv.sim (privacy), obv.sim (obliviousness), and aut (authenticity) security definitions in BHR, which we briefly describe here:

- prv.sim (privacy): There is a simulator $\mathcal{S}$ that takes as input $(1^k, f, f(x))$ and produces output which is indistinguishable from (F, X, r) generated normally.
- obv.sim (obliviousness): There is a simulator $\mathcal{S}$ that takes as input $(1^k, f)$ and

[2]In BHR's original definition, only Gb is probabilistic, while the rest are deterministic. In their syntax, there are two more notations e (encoding information) and d (decoding information). For all the efficient garbling schemes known, both e and d can be generated by a single random coin together with some associate data.

produces output which is indistinguishable from (F, X) generated normally.

- aut (authenticity): Any adversary should not be able to generate a $Y' \neq \mathsf{Ev}(\mathsf{F}, X)$ such that $\mathsf{De}(r, Y') \neq \perp$.

2.5.2 PwoP Design

The system presented in this section deals with the scenario where clients are semi-honest, the server is malicious, a fraction of sensors are malicious, and the server should be non-colluding with any other participants. We deal with the scenario with malicious clients in Sec. 2.9.

PwoP with no liveness. The general idea behind PwoP is as follows: The client is responsible for generating a garbled circuit; then sensors contribute garbled inputs; and finally the server evaluates the function using the garbled inputs, and sends the client the garbled output.

Each time the client wants to obtain a fused result of sensors inputs, the client and the sensors need to agree on a *fresh*, random coin r that is used to garble the circuit and garble the inputs respectively, and they should prevent the value r from being known by the server. In the semi-honest model, we can easily achieve this by allowing the client to dictate the coin.[3] In PwoP, we assume that a client shares a symmetric, pairwise key with each sensor. A client chooses a random coin and wraps the coin using an authenticated encryption with the pairwise keys shared. The ciphertexts will be sent to the server who will distribute them to respective sensors.

[3]In the malicious model, agreeing on a common coin can be achieved by modifying a threshold coin-flipping protocol [31] to the server-aided setting; each message in this coin-setup protocol is transmitted between the client and corresponding sensor using end-to-end encryption, with the server simply passing these encrypted messages.

Setup and inputs: Let fta be any sensor averaging function in Sec. 2.4. Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ be a garbling scheme. Let $\langle\text{REQ}\rangle$ be a client request that contains the function description. Let $x_\Pi = \{x_1, \cdots, x_n\}$ and $X_\Pi = \{X_1, \cdots, X_n\}$ be sensors' inputs and garbled inputs respectively.

00 Client c selects a random coin r, runs Gb (using r) to garble a circuit F for fta, and sends F, $\langle\text{REQ}\rangle$, encrypted coins to server S.
01 S forwards $\langle\text{REQ}\rangle$ and the corresponding encrypted coins to sensors.
02 Sensors Π run En (using r and x_Π) and send S garbled inputs X_Π.
03 Server S runs Ev on X_Π and sends the garbled output Y to c.
04 Client c runs De (using r and Y) to get $\mathsf{fta}(x_\Pi)$.

FIGURE 2.3: PwoP with no liveness. When the server receives a garbled circuit, the server collects data from sensors and returns the reply to the client.

Alternatively, we can assume public key infrastructure and our system can be easily adapted.

Also, the client can send both the wrapped coins and the garbled circuit in the same round, saving one communication round.

The above approach opportunistically leverages the bandwidth and computation asymmetry property in the sensor fusion setting, where the bandwidth between clients and the server is ample, but the bandwidth between sensors and the server is limited. In fact, it is very common in modern systems to shift part of work to clients to improve the service throughput and reduce the latency. As we will show, the overhead incurred by the circuit generation and the circuit transmission in PwoP, for practical parameters, is negligible. Moreover, in our approach, the circuit size (related to the accuracy of the returned results to clients) can be flexibly decided by clients. In addition, letting the client code allows the circuit to be precomputed off-line.

We describe PwoP with no liveness (i.e., with no guaranteed output delivery) in Figure. 2.3, using a language of garbling schemes that is slightly modified from BHR. We make black-box use of a general sensor averaging function fta, and we defer the circuit design, optimization, and justification to the next section.

We have the following theorem establishing the security of the above scheme.

Theorem 1. *The PwoP protocol achieves Privacy and Correctness for sensor fusion function* fta.

Privacy follows from both privacy and obliviousness of the garbling scheme. Specifically, the first part of Privacy (i.e., the client only learns the aggregated value) can be derived from the privacy of the garbling scheme. This is because in PwoP the client only obtains (F, Y, r), while adversary in the privacy game of the garbling scheme obtains (F, X, r) and Y is fully determined by X and F. The second part of Privacy can be trivially obtained from the privacy of the garbling scheme. Correctness follows from the authenticity of the garbling scheme and the correctness of fta. Note that correctness holds even against a malicious server as long as the server does not collude with sensors.

However, the scheme in Figure. 2.3 does not achieve liveness: if some of these sensors fail to provide their garbled inputs, the server cannot evaluate the circuit.

Supporting general feedback function. In PwoP, after the server evaluates the garbled circuit, it can also send the garbled outputs to sensors which may run De to get the fused value. In a control program, the data sent to sensors which may be co-located with actuators are *feedback data*, according to which actuators can perform some prescribed operations.

Moreover, PwoP can be easily extended to the case where S provides sensors with output from an *arbitrary* feedback function (not necessarily the same function which the client asks to compute). To achieve this, clients not only garble the function that they need but also the feedback function for sensors and actuators.

Discussion. This basic scheme shares some similarities with both Feige, Kilian, and

Naor (FKN) [67] and Kamara, Mohassel, and Raykova (KMR) [101]. The difference is that FKN and KMR only involve a server and parties (in our case, sensors) and the server needs to send back the garbled output to the parties, while in our model, the server needs to return the garbled output to the client and *optionally* to the sensors. In FKN and KMR, the server and one party do heavy work that is linear in the size of the circuit, while in our case, each sensor's role is symmetric and each sensor only does work that is linear in the size of its input. The security of KMR requires only obliviousness and authenticity of the garbling scheme, while PwoP additionally requires privacy of the garbling scheme.

Our scheme is also similar to Naor, Pinkas, and Sumner (NPS) [154], one designed specifically for auctions. In NPS, there is an auction issuer who generates the circuit, a number of bidders who send their garbled values, and an auctioneer who computes the garbled values and returns the final result to all bidders. Instead of relying on an external, trusted circuit issuer, our circuit generator is just one participating client (who would also expect a reply from the server). Moreover, NPS uses proxy oblivious transfer to provide the parties with the garbled input, but we choose to use an agreed common coin, just as FKN and KMR, for the purpose of efficiency and scalability.

The servers in both FKN and NPS can learn the output, while KMR and ours do not.

2.5.3 Achieving Liveness

We now describe PwoP with liveness. In our approach, the absence of a reply from a sensor will be treated as an input of $[-\infty, +\infty]$ (or the prescribed upper and lower bounds), which means this reply will not be counted. The reason why we can do

Setup and inputs: Let fta be any sensor averaging function in Sec. 2.4. Let $\mathcal{G} = (\mathsf{Gb}, \mathsf{En}, \mathsf{Ev}, \mathsf{De})$ be a garbling scheme. Let $\langle \text{REQ} \rangle$ be a client request that contains the function description. Let $x_\Pi = \{x_1, \cdots, x_n\}$ and $X_\Pi = \{X_1, \cdots, X_n\}$ be sensors' inputs and garbled inputs respectively.

```
00  Client $c$ selects a random coin $r$, runs $\mathsf{Gb}$ (using $r$) to garble a circuit $\mathsf{F}$ for $\mathsf{fta}$, and sends $\mathsf{F}$, $\langle \text{REQ} \rangle$, encrypted coins to server $S$.
01  $S$ forwards $\langle \text{REQ} \rangle$ and the corresponding encrypted coins to sensors.
02  Sensors $\Pi$ run $\mathsf{En}$ (using $r$ and $x_\Pi$) and send $S$ garbled inputs $X_\Pi$.
If $S$ does not receive all the garbled inputs before the times expires, it requests from the client missing garbled inputs that encode $[-\infty, +\infty]$.
04  Server $S$ runs $\mathsf{Ev}$ on $X_\Pi$ and sends the garbled output $Y$ to $c$.
05  Client $c$ runs $\mathsf{De}$ (using $r$ and $Y$) to get $\mathsf{fta}(x_\Pi)$.
```

FIGURE 2.4: PwoP with liveness. The protocol completes with 1 round in the failure-free scenario, and with 2 rounds if some garbled inputs are missing.

this is that our fault-tolerant algorithms can natively tolerate empty (meaningless) inputs as long as the number of these inputs (and together with malicious inputs) are g-bounded. More specifically, if the server does not receive the garbled input from some sensors in time, it will ask the client to send corresponding garbled inputs for the missing sensors for values $[-\infty, +\infty]$. When using algorithms with bounded accuracy, the client will generate a random interval with maximum accuracy. PwoP with liveness is described in Figure. 2.4.

We thus have the following theorem.

Theorem 2. *PwoP described in Figure. 2.4 implements Liveness in synchronous environments.*

In synchronous environments, if the server is correct and if all Byzantine sensors crash, the server will request garbled inputs from the correct client after the timer expires. As the client is correct, the server will receive these "dummy" garbled inputs and the server can evaluate the garbled circuit and send a reply to the client.

Discussion. Achieving liveness *efficiently* has been a difficult problem for garbled circuit based multi-party computation. Most of prior works on GC (as surveyed in Sec. 4.2) achieve liveness by introducing multiple servers and use secret-sharing based techniques to help liveness. This is not only less efficient in general, but also requires significant communication and interaction, which makes it ill-suited in bandwidth and energy restricted environments.

Schemes in [101, 103] do not achieve liveness. NPS [154] considered a "denial of service attack by bidders," which is essentially a form of liveness. Similar to our case, NPS needs to prevent a corrupt bidder from sending incorrect values or simply not sending values to the server. Their approach is to prove to the auction issuer that the bad event occurs, and then a dummy zero value will be provided. However, in our case we aimed at minimizing the interaction with the client (circuit generator). In addition, our scheme achieves liveness for random coin based approach, while NPS uses proxy oblivious transfer.

A similar problem was studied by Feigenbaum, Pinkas, Ryger, and Saint-Jean [68]. They simply provided a solution that requires all the parties to pre-commit their values to two "non-colluding" servers *before* a circuit is garbled. Their application scenarios are very different from ours.

In PwoP, malicious sensors may contribute ill-formed garbled inputs so that the server ends up with outputting $\perp$. It is vital that the server can "quickly" tell if a garbled input is correct. We discuss how PwoP can achieve fast detection for ill-formed inputs in Sec. 2.9. This way, PwoP can be extended to achieve liveness for g-bounded Byzantine sensors.

2.6 Circuit Design and Optimizations

This section describes how to design *efficient* circuits for the fault-tolerant algorithms in Sec. 2.4. There are three good reasons why we need the effort.

First, while multiple generic GC compilers or tools that can translate a program to a circuit exist [134, 177, 85, 18, 200, 120], there is significant room for improvement for some specific programs. Our optimization requires non-trivial efforts and analysis for the correctness of the design.

Second, among all the GC compilers, TinyGarble [177] is generally deemed to be (one of) the most efficient one, especially for large programs, as it incorporates state-of-the-art optimizations such as free-XOR [118], row reduction [154], fixed-key AES garbling [18], and half gates [201], and more importantly, uses *logic synthesis* to reduce the size of circuits. However, TinyGarble only supports a limited number of components that we need. We therefore aim to build modular components that can be readily used for our fault-tolerant algorithms.

Third, different from the conventional circuit design, the sensors in our setting can be malicious and may contribute malicious garbled inputs. Garbled circuits, or in general, multiple-party computation, offer no protection on malicious inputs. For instance, we cannot rely on sensors to contribute well-formed input intervals (see below).

Convention. Before the protocol starts, all participants should agree on a representation for the intervals and d-dimensional values. For intervals, each possible value is given an integer label with l bits, and we assume the lower and upper bounds are 0 and σ respectively. Clearly, $l = \log \sigma$. Likewise, d-dimensional values are given a d-dimensional vector with each component being an interval which can be represented by a fixed number of bits.

2.6.1 Circuit design for M-g-U

Overview. Our complete circuit design for M-g-U is depicted in Figure. 2.5. To implement Marzullo's algorithm, we first need to sort all the input endpoints of sensor inputs, resulting in a sorted array of $2n$ values (considering each sensor is providing one interval in the form of two endpoints). This is achieved using *modified sorting networks.* For each point in this sorted array, we need to count how many input intervals can cover this point. This is handled by adding 1 to an intersecting interval counter if the point is a left end of an input interval, and subtracting 1 if it is a right end. After that, we compare the intersecting interval counters for each point with $n - g$, in order to find the points that are covered by *exact* $n - g$ intervals. We do this using *index select.* As in Figure. 2.5, the left end of the resulting interval is the output of the *max value min index* module.

Likewise, the circuit for computing the right end of the resulting interval can be implemented in a symmetric way by again running the modified sorting networks and index select. However, we will show that we can *reuse* the modified sorting networks and index select module for computing the right interval, as shown in Figure. 2.5.

Our circuit design in detail. Instead of using $(+1, u)$ and $(-1, v)$ to represent an interval $[u, v]$ (as in the code version of Marzullo's algorithm), in our circuit design, each sensor provides an interval in the form of two values u, v to the server. This is because sensors may be malicious and it will result in wrong result if the left end provided by some malicious sensor is actually larger than the right end. Therefore, at the first level of our modified sorting network, we need to add an array of compare-

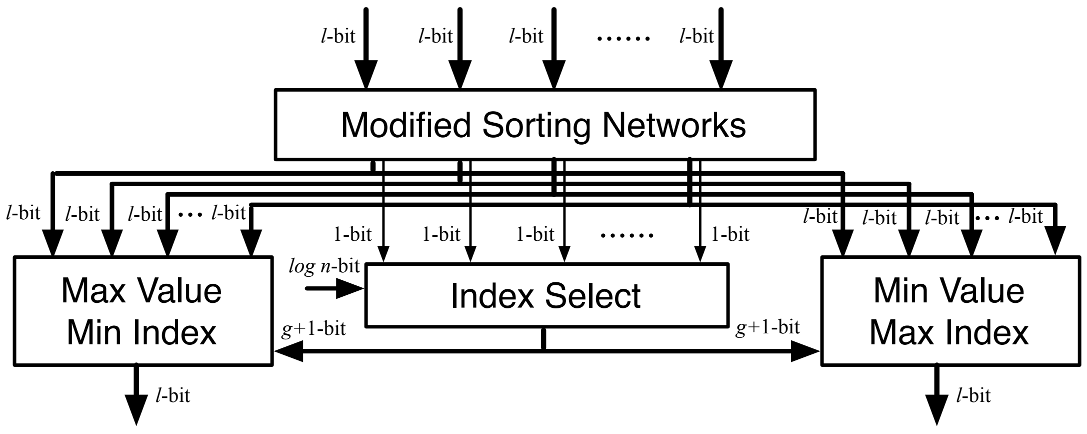


FIGURE 2.5: The complete circuit design for M-g-U. n is the number of sensors, l is the length of the endpoint of the input interval, and g is the number of faulty sensors.

and-swap modules to sort the two values from the same sensor. Note that the above problem is not what the conventional garbled circuit design would care about.

Before we proceed, let's recall sorting networks [15, 46, 114], which are circuits that sort a sequence into a monotonically increasing sequence. The core building block of a sorting network is a compare-exchange circuit, which takes as input a pair of values (x, y) and returns a sorted pair (x', y') so that $x' = min(x, y)$ and $y' = max(x, y)$. To realize this building block, prior constructions [118, 117, 84] used the idea of *compare then conditional-swap:* the circuit keeps two inputs unchanged if and only if the comparator returns 1, i.e., x is less than y. For our design, the first layer of our modified sorting network guarantees that the two values from the same sensor will form an interval with its right end always greater or equal to its left end. Then we taint these two values with $+1$ and -1 respectively to indicate the order of two endpoints. For our implementation, we use one bit to represent for ± 1, i.e., we use "1" and "0" to represent $+1$ and -1 respectively.

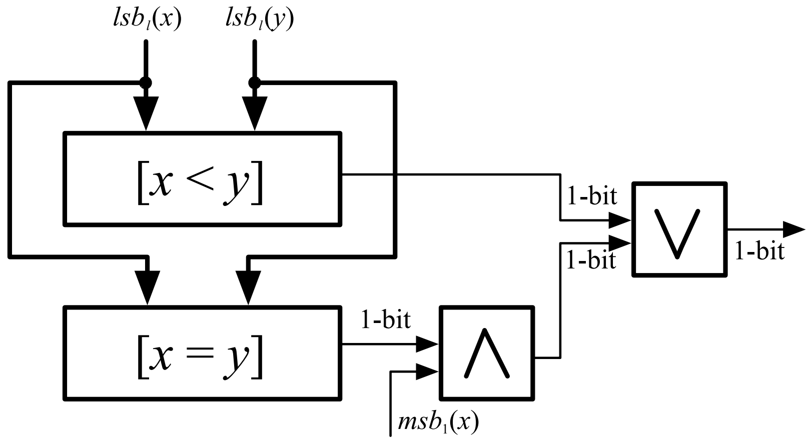


FIGURE 2.6: The comparator component.

Starting from the second layer of our modified sorting networks, it is essentially sorting networks built from compare-exchange components that have a modified comparator, as illustrated in Figure. 2.6. Instead of using the *less-than* comparator, we follow Marzullo's algorithm [142] to realize a comparator, $<_m$, as defined below: Given two inputs x and y, each of which is of the form $s||u$ where $s \in \{1, 0\}$ and $|u| = l$, define $x <_m y = (lsb_l(x) < lsb_l(y)) \vee (lsb_l(x) = lsb_l(y) \wedge msb_1(x) > msb_1(y))$, where lsb_l and msb_l represent the least significant l bits and the most significant l bits respectively. In words, $x <_m y$ if and only if the value part of x is less than that of y, or the value parts happens to be equal and the sign part of x is greater than that of y. Note that the sign part of a left end is encoded by 1.

We follow [117] to realize the conventional less-than circuit and equal-to circuit (which takes advantage of the free-xor technique). However, observing that when $lsb_l(x) = lsb_l(y)$ and $msb_1(x) = 1$ it does not matter we swap or not the two inputs, we can further simplify the circuit, leading to a circuit exactly as in Figure. 2.6.

For our implementation, while asymptotically optimal sorting network exists [4], PwoP uses Batcher sorting network [15, 114] which has much better performance for practical parameters, as studied in [134].

Now we describe how to find the position of the minimum value of the resulting interval by our index select module composed by a prefix sum circuit and an array of equality checkers, as shown in Figure. 2.7. All the sorted one-bit inputs, representing $+1$ or -1, first go through a prefix sum circuit to compute their prefix sums. Prefix sum circuit allows one to compute on input $(z_1, z_2, \cdots, z_n)$ and produce as output $(m_1, m_2, \cdots, m_n)$, where $m_j = z_1 + z_2 + \cdots + z_j$ for $1 \leq j \leq n$. Indeed, this circuit fits perfectly for our purpose as the intersecting interval counter in M-g-U. A straightforward instantiation of n-prefix sum circuit requires n additions.[4] Note that for each addition when performing prefix sums, we can exploit the free-xor circuit from [117].

The next layer is to convert every prefix sum which is equal to the value $n - g$ to 1 and convert the rest to 0 otherwise. This can be trivially instantiated via simple equal-to circuit [117]. Observing that not every position in the array of prefix sum can possibly equal $n - g$, we can apply another optimization that only implements comparators for the positions where $n - g$ can be the possible output. To be precise, for an array with $2n$ values provided by n sensors, only $g + 1$ positions can possibly have a prefix sum equal $n - g$. We prove the correctness of this optimization in Theorem 5 in Appendix B.1. This observation reduces the number of comparators and the width of max value min index by roughly 83.3% of that with a straightforward implementation, because it only compares at $g + 1$ positions, instead of $2n = 6g + 2$ positions in a straightforward implementation. However, the size reduction for the

[4]In a system that can evaluate garbled circuits in parallel, we recommend to implement a parallel prefix sum circuit as mentioned in (cf. [165, Chapter 2.6]), which has a depth of $O(\log n)$ and $O(n)$ additions.

other algorithms is slightly smaller, with a 75% reduction, since $n = 2g + 1$ for the other algorithms.

Then these $g+1$ values go through a max value min index circuit which computes the value with the minimum index and the maximum value (which is 1). Effectively, it outputs the leftmost point which covers $n-g$ sensor input intervals. We can easily modify the circuit from [117] to obtain this circuit.

To compute the maximum value, we find that one can reuse the result of modified sorting networks and index select modules. Specifically, we just need to left shift one position of the sorted input value array (shifting is free in circuits), and apply it to a max value max index circuit. Then we will generate the right end of the result interval. The proof of correctness of this optimization is in Theorem 6 in Appendix B.1. Since the whole circuit complexity is dominated by the sorting network, this optimization avoids another sorting networks and index select circuit for computing the right end of the resulting interval, thereby halving the computation overhead.

Caveat. A tempting way of designing the circuit for M-g-U might be to first use the sorting network and then "release" all the flag information, which the server could use to easily determine the indexes and the final result. One might think that this flag information would not reveal much information, or one might be willing to trade this information for efficiency and the compactness of the circuit. However, we are not sympathetic to this viewpoint, stressing that the flag information leaks too much to the server. Indeed, the server can tell from the flag information the topology of all the intervals, and even tell the number of failed sensors. Also note that only sorting the entries (without giving the final interval) would not lead to a garbling scheme that achieves authenticity when the server is malicious.

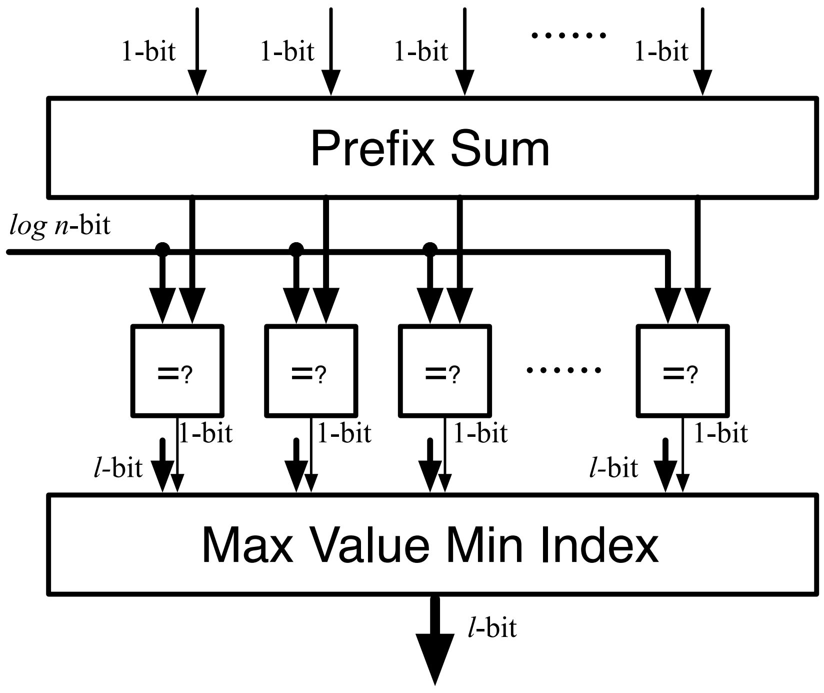


FIGURE 2.7: The index select module and the max value min index module. The index select module is composed by a prefix sum circuit and an array of equality checkers.

2.6.2 Circuit Design for Other Algorithms

Circuit design for M-g. M-g is a sensor averaging algorithm for arbitrary failures with *bounded* accuracy. Recall that our convention is that each interval is of the range $[0, \sigma]$, where σ is some pre-determined, maximum value. Here we need to levy an additional requirement on the input interval $[u, v]$ (where $0 \leq u \leq v \leq \sigma$): the difference between u and v must be less than or equal to some threshold t, i.e., $v - u \leq t$; otherwise the interval is deemed as being “invalid.”

Circuit design for M-g-m. Our circuit for M-g-m builds on that of M-g. We calculate the midpoint of the resulting interval by adding the two end points together,

Table 2.1: PwoP characteristics. The column labeled "type" specifies if the algorithm handles intervals (I) or d-dimensional values (dD). n is the total number of sensors, g is the upper bound on the number of malicious sensors, k is the security parameter (in this paper, 128 bit), and l is the length of sensor input. The columns labeled "circuit size", "comp", and "comm" specify for server circuit complexity, sensor computation time complexity (measured using the number of pseudorandom function calls), sensor communication complexity for PwoP, respectively.

algorithms	type	#sensors	description	circuit size	comp	comm
M-g-U	I	$3g+1$	unbounded accuracy	$O(ln\log^2(n))$	$O(l)$	$O(lk)$
M-g	I	$2g+1$	bounded accuracy	$O(ln\log^2(n))$	$O(l)$	$O(lk)$
M-g-m	I	$2g+1$	only reveal midpoint	$O(ln\log^2(n))$	$O(l)$	$O(lk)$
M-op	I	$2g+1$	"optimistic"	$O(ln\log^2(n))$	$O(l)$	$O(lk)$
SS	I	$2g+1$	Lipschitz condition	$O(ln\log^2(n))$	$O(l)$	$O(lk)$
ChM-dD	dD	$2dg+1$	d-dimension	$O(ldn\log^2(n))$	$O(dl)$	$O(dlk)$
ChM-dD-sso	dD	$2g+1$	same size & orientation	$O(ldn\log^2(n))$	$O(dl)$	$O(dlk)$

without dividing it by 2. Since it is a division by 2 (a public constant), users can divide it by themselves without using garbled circuits.

Circuit design for M-op. Recall that M-op wishes to find the cover for the maximum intersection groups for all the intervals. Our circuit for M-op is similar to that of M-g-U. The difference is that no equality checkers are needed in index select, since g is unknown. Also, the comparators in max value min (max) index will need to be integer comparators, instead of binary comparators, because we are looking for the points that have the highest prefix sum, which implies that these points intersect with the maximal number of input intervals.

However, it is worth noting that some of the optimizations for M-g-U circuit cannot be applied here, because g is unknown to M-op algorithm. Specifically, we are unable to select only $g+1$ values for comparisons and selections, we have to feed all the values into the max value min (max) index modules.

Circuit design for SS. The core functionality of SS is selection, with which one can easily derive the final interval. To realize the selection circuit, one solution is to directly apply a conventional sorting circuit on the input values, and we can easily obtain a circuit with $O(ln \log^2 n)$ complexity. Another possibility is to use the probabilistic method from Wang et al. [187]. This gives a non-zero error probability, but leads to a circuit with $O(ln \log k)$ complexity. In PwoP, we implement only the method with zero error probability.

Circuit design for ChM-dD. One can project the region for sensors onto each of the d-dimensions, and thus obtain d independent 1-dimensional problems, which can be resolved separately. The combined rectangles are called *projection rectangles*, and this approach is called the *projection approach*. It is easy to see that since the projection algorithm naturally gives rise to a d-rectangle, no further adjustments on the shape of the rectangles are needed.

We can generalize our M-g circuit to handle the case of d-dimensional projection rectangles in a straightforward manner: d independent circuits will be generated and the size for ChM-dD is just d times as large as that of M-g.

According to Chew and Marzullo [45], the projection approach for ChM-dD is "the method-of-choice for some situations," but there are two disadvantages to the approach [45, Section 3.1]: first, some information may be lost; second, the size of the average result may be larger than necessary.

Circuit design for ChM-dD-sso. ChM-dD-sso is referred to as the algorithm finding the fault-tolerant rectangle from ones with the same size and orientation. In this case, one may simply compute the projection d-rectangles in $O(dn \log n)$ time tolerating $g \leq n/2$ failures. Note that the maximum tolerable failures g is independent of the number of dimensions d. We note that the combined rectangle may be smaller than the original size. For our design, we do not attempt to construct a rectangle with exactly the same size as hinted by [45], and instead we simply provide the user with the minimum one and the user can easily construct a rectangle from the result.

Circuit design for d-rectangles with unbounded accuracy. For ChM-dD and ChM-dD-sso, there are variants for arbitrary failures with unbounded accuracies. Their circuits can be constructed following an analogous line as what we did for the case of M-g-U and M-g.

2.7 PwoP Characteristics

Table 2.1 shows the characteristics of PwoP. PwoP covers seven Byzantine fault-tolerant sensor fusion algorithms. PwoP is designed specifically for cyber-physical systems that work in computation and bandwidth restricted environments.

In PwoP, the communication round is optimal: one round for failure-free scenarios, and two rounds for failure scenarios. Meanwhile, sensors performs "minimal" computation, depending on l (and d) only. The total bits that each sensor sends to the server only depend on l and the security parameter k (and d). These metrics in PwoP outperform the ones in prior privacy-preserving sensor aggregation systems with pollution attacks mitigated such as [47, 124] and an alternative approach using set representation described in Appendix A.1.

2.8 Implementation and Evaluation

2.8.1 Implementation

We make a general garbled circuit compiler specifically for our clients-server-sensors setting. We achieve this by modifying and extending TinyGarble [177], the state-of-the-art garbled circuit compiler for secure two-party computation, which incorporates state-of-the-art garbled circuit optimizations [118, 154, 18, 201] and leverages logic synthesis to optimize and compress circuits. Specifically, we first decompose TinyGarble into three parties: clients (garbled circuit generators), the server (garbled circuit evaluator), and sensors (garbled input providers). We then modify the system to be coin-based: clients and sensors now take as input shared random coins. Last, we modify the garbled circuit evaluation function so that garbled output is hidden from the server (which is essential to achieving Privacy).

As TinyGarble does not support efficient sorting networks or other primitives we need, we directly build optimized modules needed and then build the circuits described in Sec. 2.4. We manually write the hardware circuits in Verilog, as we find doing so also provides circuit reduction compared to the ones using high-level programming languages and high level synthesis. The resulting circuits go through another logic synthesis process by Synopsys Design Compiler to obtain the netlists of the implemented algorithms. Lastly, we applied the V2SCD tool in TinyGarble to convert netlists into simple circuit description files, which can be taken as the inputs of TinyGarble framework.

To show the practicality of our system, we build a multi-sensor fusion system using Raspberry Pi Zero W (1GHz, single core CPU and 512MB RAM). In our system, each Raspberry Pi functions as a sensor, providing the server with garbled inputs for its sensor inputs. Raspberry Pi Zero W is the cheapest Raspberry Pi device supporting WiFi connection, and is smaller than a credit card. Notice that, we selected Raspberry Pi Zero W just because of its built-in WiFi module, not its computational power. As a fact, the computation on each sensor only contains a few pseudorandom function evaluations and XOR operations, so it is very efficient for any sensor with minimum computational power to compute. The server runs on an Intel Core i7-4790 processor, and the client runs on another computer with an Intel Core i7-4702HQ processor. Sensors and the server are connected using one wireless router. To support concurrent transmissions of sensor inputs, we implement multi-threading for data collection at the server side. The client and the server are physically connected via an Ethernet cable.

The length of all sensor values (i.e., $l = \log(\sigma)$) is set to eight bits. For an interval, each sensor needs to garble a 16-bit input, resulting in a 256-byte garbled input. Similarly, garble inputs for a rectangle in our setting takes 512 bytes.

Our system achieves liveness for sensors that are subject to crash failures. As described in Sec. 2.5, our system works in synchronous environments and needs a two-round communication when there are sensor failures.

Initially, PwoP uses heartbeat messages to detect failed sensors before the sensor fusion protocol starts, and sets a timer for each request to detect new crash faulty sensors. Specifically, the server sets a timer each time a request is sent to sensors. If it does not receive some garbled inputs in time, it marks the corresponding sensors as faulty. It then asks the client to provide garbled inputs for $[-\infty, +\infty]$ (or random values in the case of algorithms with bounded accuracy) for these sensors, evaluate the circuit, and send the garbled output to the client.

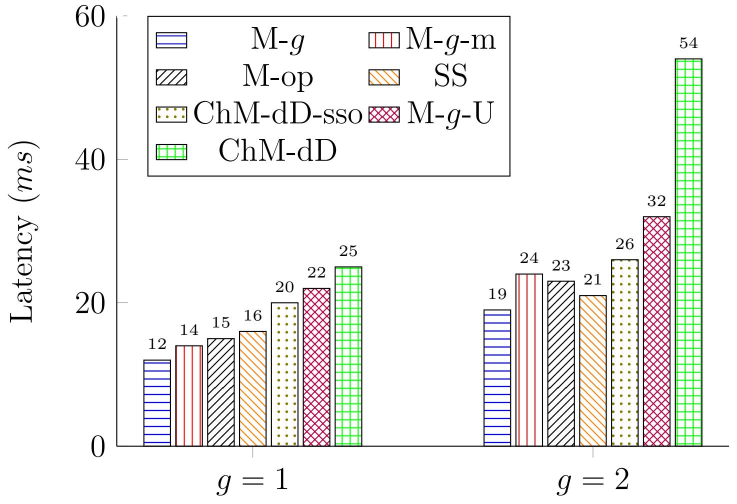


FIGURE 2.8: Latency (in ms) of PwoP in failure-free scenarios for $g = 1$ and 2. This and subsequent figures are best viewed in color.

2.8.2 Evaluation

Overview. We have evaluated PwoP using up to 19 sensors, with seven different algorithms: M-g, M-g-m, M-g-U, M-op, SS, ChM-dD, and ChM-dD-sso. We evaluate PwoP for these algorithms under different network sizes (the number of sensors). We use g to represent the network size for these algorithms, and total number of sensors can be found in Table 2.1. We measure the latency, throughput, and scalability in both failure and failure-free scenarios. For failure-free scenarios, each sensor will provide a well-formed garbled input, even if the underlying value is faulty. In contrast, our failure scenario captures crash failures where some sensors do not provide the server with garbled inputs in time.

Overall, PwoP has low latency and high throughput, and can be deployed in many real-time applications, e.g., monitoring pressure and water leaks in a water distribution system [6].

Latency. The latency evaluation for the seven algorithms are depicted in Figure. 2.8 for $g = 1$ and 2. The latency of the whole process takes only 12 to 54 milliseconds.

To understand the performance bottleneck of the system, we look into the time consumed by each operation for the whole process. We find that the communication between the server and sensors is two orders of magnitude larger than the cryptographic operations and the rest of the communication. Circuit generation (at the client side), circuit transfer (between the client and the server), garbled input generation (at the sensor side), and circuit evaluation (at the server side), collectively, take only several milliseconds, even when $g = 9$ and the total number of sensors is 19. This implies that if we can reduce the communication latency between the server and sensors, we can easily boost the performance of PwoP.

To compare the performance of different algorithms, we notice that for a given g, M-g, M-g-m, M-op, and SS (the four one-dimensional algorithms) have roughly the same latency, because the latency is dominated by the communication time between the server and sensors, and the total size of garbled inputs transmitted for these algorithms is exactly the same. This observation is less visible small g's (see Figure. 2.8), but becomes apparent for larger g's (see Figure. 2.10).

For the same g, the latency of M-g-U is larger than that of the other one-dimensional algorithms. Indeed, to tolerate g sensor failures, M-g-U requires more sensors ($3g+1$) than the other one-dimensional algorithms ($2g+1$). As a consequence, it requires more data to be transmitted, and takes longer time to process all the data.

Besides these one-dimensional algorithms, we implement and evaluate d-dimensional algorithms. In particular, we set $d = 2$ for our evaluation, as 2-dimensional sensor fusion is useful for many applications (e.g., measuring the location of a physical object). The size of each sensor endpoint is set to 8 bits, and a two-dimensional rectangle re-

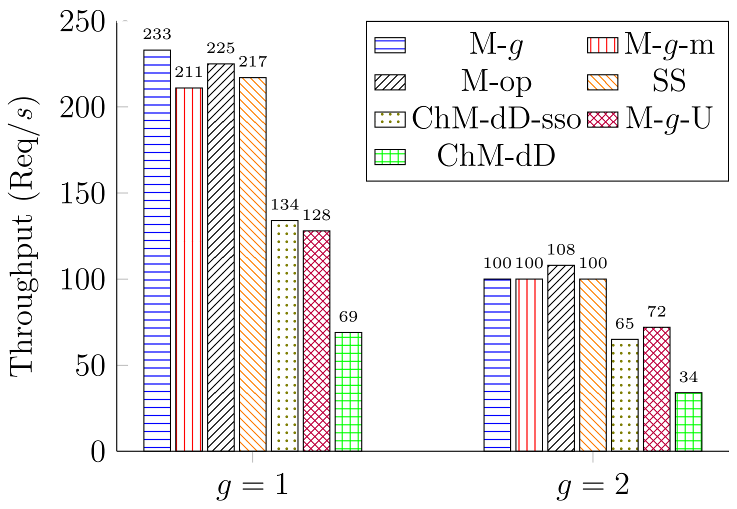


FIGURE 2.9: Throughput of PwoP in failure-free scenarios for $g = 1$ and 2.

quires 32 bits to represent. This doubles the size of the garbled inputs, comparing to the case for one dimensional algorithms. We find that the increased data size is reflected in latency: the latency of ChM-dD-sso is larger than that of M-g, M-g-m, M-op, and SS for the same number of sensors. ChM-dD requires $2dg+1$ to tolerant g malicious sensors, and therefore in the 2-D example $4g+1$ sensors are required. Correspondingly, the latency of ChM-dD grows much faster than the other algorithms.

Throughput. The throughput of PwoP for all seven algorithms for $g = 1$ and 2 is shown in Figure. 2.9. Similar to the latency analysis, the throughput of M-g, M-g-m, M-op, and SS is almost the same for the same g values. The throughput of M-g-U is consistently lower than the other one-dimensional algorithms for the same g, due to a larger number of connected sensors. The throughput of ChM-dD-sso is lower than all one-dimensional algorithms because there are more transmitted input data for ChM-dD-sso. Moreover, the throughput of ChM-dD is even lower due to even more sensors in total.

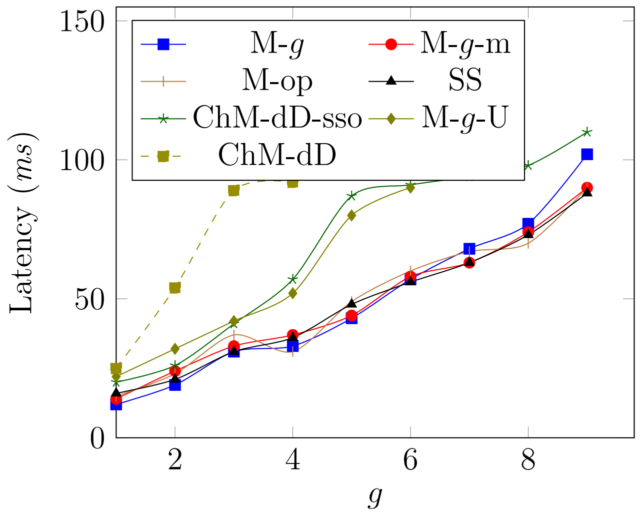


FIGURE 2.10: Scalability of the latency of PwoP in failure-free scenarios for $g = 1$ to 9. (except M-g-U and ChM-dD, for which we measured for $g = 1$ to 6 and $g = 1$ to 4, respectively)

In PwoP, we did not implement batching of sensor inputs for the communication between the server and sensors. We conjecture that the use of batching would provide higher throughput and a meaningful trade-off between latency and throughput. We did not implement parallel evaluation of garbled circuits, and this is not needed in our current implementation for which the communication is the bottleneck.

Scalability. We evaluate the scalability of PwoP using up to 19 sensors. The largest latency we obtained in our evaluation is the latency of ChM-dD-sso for $g = 9$, which is only 0.11 seconds, as shown in Figure. 2.10. This demonstrates that PwoP, for all algorithms and all network sizes that we tested, is efficient. In Figure. 2.10, as the number of sensors increases from 3 to 19, the latency grows slowly and linearly from 0.012 to 0.11 seconds.

If we compare the latency between M-g-U and the other algorithms for the case where the total number of sensors are equal, e.g., $g = 6$ for M-g-U and $g = 9$ for

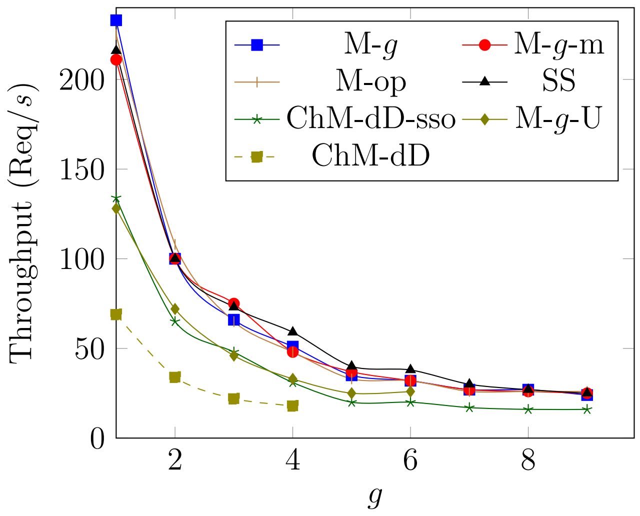


FIGURE 2.11: Scalability of the throughput of PwoP in failure-free scenarios for $g = 1$ to 9. (except M-g-U and ChM-dD, for which we measured for $g = 1$ to 6 and $g = 1$ to 4, respectively)

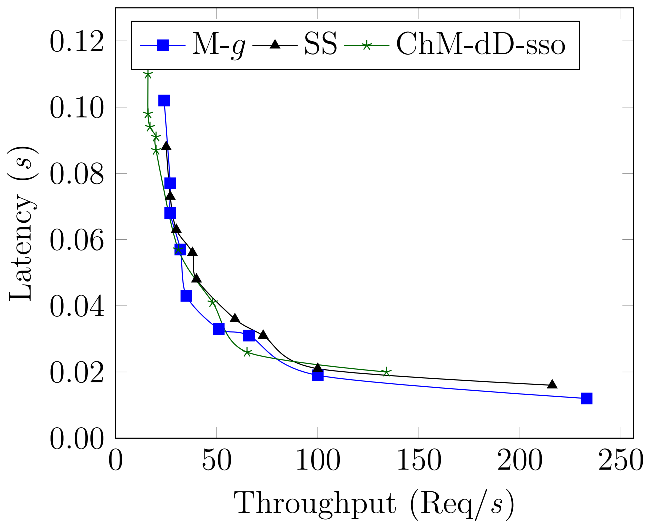


FIGURE 2.12: Latency vs throughput of PwoP in failure-free scenarios.

M-g-m, the latency difference is almost not noticeable. This, from a different angle, confirms that the latency is dominated by the communication, not the cryptographic operations.

In addition, the scalability of throughput of PwoP is shown in Figure. 2.11. As

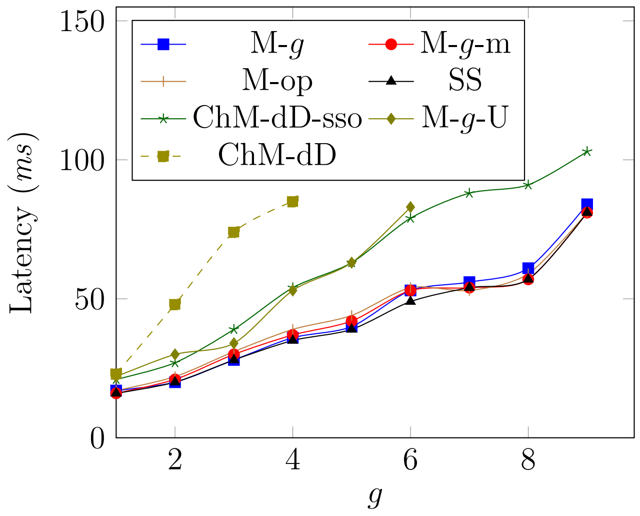


FIGURE 2.13: Scalability of the latency of PwoP in failure scenarios for $g = 1$ to 9 (except M-g-U and ChM-dD, for which we measured for $g = 1$ to 6 and $g = 1$ to 4, respectively)

the number of sensors increases, the throughput reduces from 233 Req/s to 24 Req/s for M-g, and similarly for others. This is due to the network congestion incurred by the increase of the number of sensors.

We also report latency vs. throughput of three algorithms M-g, SS and ChM-dD-sso in Figure. 2.12. For clarity, we omit the rest as the curves of the other algorithms are very similar to these three.

Performance in failure scenarios. In failure scenarios, we test the case where g out of n sensors fail at the same time. The latency and the throughput for PwoP for $g = 1$ to 9 in failure scenarios are shown in Figure. 2.13 and Figure. 2.14, respectively.

The latency part does not count the time-out value set by the server. It should be set to different values for different applications. We find that there is no observable difference between the failure and failure-free scenarios for latency if not counting the time-out value. Indeed, the extra communication does not add visible overhead.

We also find that the throughput in failure scenarios are slightly better than the

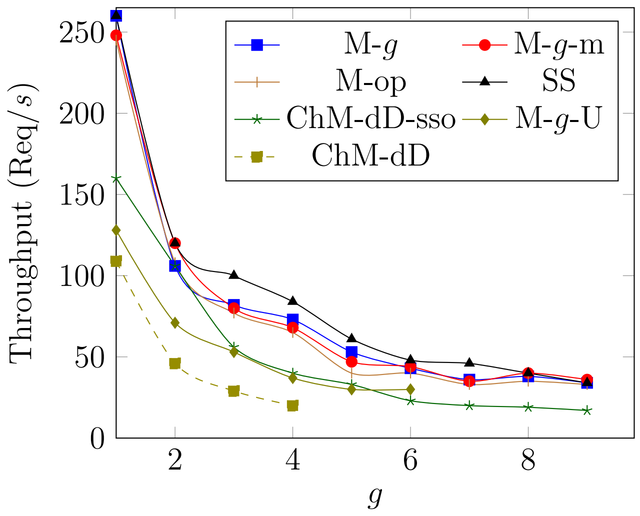


FIGURE 2.14: Scalability of the throughput of PwoP in failure scenarios for $g = 1$ to 9. (except M-g-U and ChM-dD, for which we measured for $g = 1$ to 6 and $g = 1$ to 4, respectively)

that in failure-free cases. The reason is that comparing with getting garbled inputs from sensors via WiFi, it is faster to get garbled inputs from the client, which is connected with the server using an Ethernet cable physically. Moreover, as now there are less sensors competing for the bandwidth between sensors and the server, their communication is faster.

2.9 Discussion

PwoP with malicious clients. We have studied how to design schemes with semi-honest clients. Indeed, clients have incentives to learn correct results from the server. However, malicious clients can learn information which should be kept private. In particular, malicious clients might code a circuit that computes functions that are different from the one as claimed by clients. For instance, clients may code circuits on computing how many and which sensors have failed, or a circuit that reveals (some) sensors' individual inputs.

It is easy to secure our system against malicious clients. This is (easily) achieved by augmenting and modifying the Salus protocol due to Mamara, Mohassel, and Riva [103]. As PwoP, we require the clients to garble the circuit (instead of one of the parties), and we let the client and the server run a cut-and-choose protocol. To reduce the communication overhead, we may use the optimal strategy of Shelat and Shen [171] that challenges 3/5 of the circuits, and may use the random seed checking technique due to Goyal, Mohassel, and Smith [74].

False garbled inputs detection. We describe how the GC evaluator (the server, in our case) can efficiently detect if garbled inputs are valid.

Recall that malicious sensors may provide the evaluator with invalid garbled inputs. Depending on which garbling schemes are used, an invalid garbled input may cause the evaluator to return $\perp$ (before all the gates are evaluated), or return an invalid garbled output that appears correct to the evaluator but will not be accepted by the client.

Our goal is to detect false garbled inputs in an as efficient way as possible. The technique can be used for PwoP to achieve Liveness even against fully Byzantine sensors. To this end, we first need a garbling scheme with the underlying encryption being able to detect false garbled inputs. Ideally, we require the evaluator to perform a minimum number of circuit gate evaluation, preferably, only on circuit input gates.

Our notion of security for the underlying encryption scheme is similar to that of Lindell and Pinkas (LP) [132], where they used a symmetric encryption scheme with an *elusive* and *efficiently verifiable* range. It allows the evaluator to tell which gate entry in a four-row table is correct. If one is unlucky, it would try all of the four entries. Most recent garbled circuit implementations chose to use the "point-and-permute" technique [16], which leads to garbling schemes that allow to decrypt exactly one entry per gate.

However, our motivation is different from LP. Here we simply require that the evaluator would return "no" if the garbled input for a single bit is not one of the required two encodings. Not surprisingly, LP does meet the simple notion above. Other (and more efficient) constructions are possible, such as using (deterministic) authenticated encryption and PRP with redundancy.

For efficiency, we may use a hybrid construction as follows: At entry level of garbled circuit, we use a garbled scheme which can efficiently detect invalid garbled inputs. When evaluating the remaining garbled circuits, we can still use the most efficient GC implementation so far. We comment that for the entry level garbling scheme, we can still use point-and-permute technique, which requires the evaluator to decrypt only one table entry — we only give malicious sensors one chance.

Additional care is needed: we need to design a circuit that has a small number of entry-level gates and encompasses all the inputs. Ideally, the verification should be run in parallel. Also, we need to consider free-XOR gates, as evaluating them does not take any encryption/decryption and they do not allow efficient detection.

Take the circuit for M-g-U as an example. We may choose to use Batcher sorting network [15], not only because it is the most efficient one, but also because it allows efficient and parallel false garbled inputs detection. Before going to the subsequent circuit, Batcher sorting network first runs $2n/2 = n$ compare-exchange operations in parallel. The entry-level circuit (with n compare-exchange operations) is rather small compared to whole sorting network circuit (with $O(n \log^2 n)$ compare-exchange operations).

2.10 Conclusion

We describe the design and implementation of PwoP, an efficient and scalable system for intrusion-tolerant and privacy-preserving multi-sensor fusion. PwoP has two distinguishing features: 1) PwoP can provably defend against pollution attacks without compromising privacy, and 2) PwoP is designed specifically to perform in computation and bandwidth restricted cyber-physical systems. To show the practicality of our approach, we build a secure multi-sensor fusion system, covering a variety of practical application scenarios.

Chapter 3

Snapshotter: Lightweight Intrusion Detection and Prevention System for Industrial Control Systems

3.1 Introduction

An industrial control system lies at the heart of industrial processes' operation and automation. The term ICS usually refers to a collection of different control systems and the associated instrumentation used for the process control purposes in almost every industrial infrastructure such as transportation, manufacturing and energy industries. Distributed control systems (DCS) and supervisory control and data acquisition (SCADA) are the most common types of ICS. Each of these is suitable for different control processes based on the complexity and desired functionality of the process. Programmable logic controllers are the major control component of such systems. In a nutshell, a PLC, by being programmable (as the name itself implies) can be used for different industrial control applications. It comprises input modules which are connected to measuring sensors and output modules which are connected to output devices, e.g., actuators. The processing unit of the PLC is in charge of continuously reading the inputs from the sensors and producing the desired outputs according to the program (*logic*) to operate the actuators. Due to widespread application of such devices in critical infrastructure, lack of security considerations in the design and lifecycle of traditional ICS, and exposure to the outside world (i.e., the Internet) because of increased connectivity through embracing the new information technologies (IT), security has become an urgent concern in such environments. In this regard, being the "holy grail of cyberwar", the National Institute of Standards and Technology (NIST) provides a detailed guidance document [180] for establishing secure ICS by identifying common threats and vulnerabilities in such systems, in addition to recommendations for security solutions and risk mitigation techniques.

However, as perfect security is unattainable [140], we consider the last stage of a cyber attack lifecycle, i.e., we assume that the adversary has already established a foothold and delivered the malware to the target host(s), exploitation phase is done successfully, and it is ready to take its final action on the attack objectives. In the context of industrial controllers, this objective could be disabling the legitimate code to run, isolating it from the real I/O and running arbitrary logic on the controller (as it happened in the Stuxnet story [126] or Triton malware[1] [181]) which all have a direct influence on the performance and output of a physical production process. Therefore, a logging scheme seems to be crucial in order to be able to verify the integrity of the code running on the controllers. In addition, considering such a strong

[1]Also known as "Trojan.Trisis", it specifically targets safety instrumented systems (a type of ICS), and deploys alternative logic to such devices for disruptive purposes

adversary, a post-compromise scheme is needed to guarantee the integrity of generated logs and alerts on the hosts. This can be achieved by means of a forward-secure logging technique [138, 168, 198, 197] which uses forward-secure integrity protection by using a fresh key for each log encryption and immediate deletion of the key after use. Note that the use of message authentication codes, digital signatures, or even message encryption by itself cannot guarantee tamper-resistance or protection against log deletion, as we assume that an adversary with full access is able to learn the signing/encryption keys and consequently forge the logs in an unnoticeable manner to disguise attack evidence and evade intrusion detection.

In this regard, by taking advantage of PillarBox [29], a tool for fast forward-secure logging, we present a lightweight host-based intrusion detection system (HIDS) called *Snapshotter* for PLC, as shown in Figure 3.1. To locate signs of potential security-related incidents, the HIDS agent installed on each PLC, logs security-related events (e.g. I/O operations) on the controller. Then, the Snapshotter periodically sends the system snapshot (i.e., the logs), in a *stealthy and fast forward-secure way* to a trusted server for the purpose of analysis and intrusion detection. To detect suspicious behaviors and operations on each PLC, the server can first check the integrity of the log itself (in case, the attacker has already compromised the device). Moreover, the validity of the program running on the controller (and consequently the I/O operations) can be verified by tracing deviations from the expected PLC profile (i.e., expected I/O behaviors) which could be established based on the legitimate logic during the system installation time. If any of the previous incidents happen, i.e., whether the log's integrity check fails, or an operation is detected as invalid, a flag will be raised and an intrusion is indeed captured. In that case, the server can take consequent actions such as further investigations of device status, recovering the

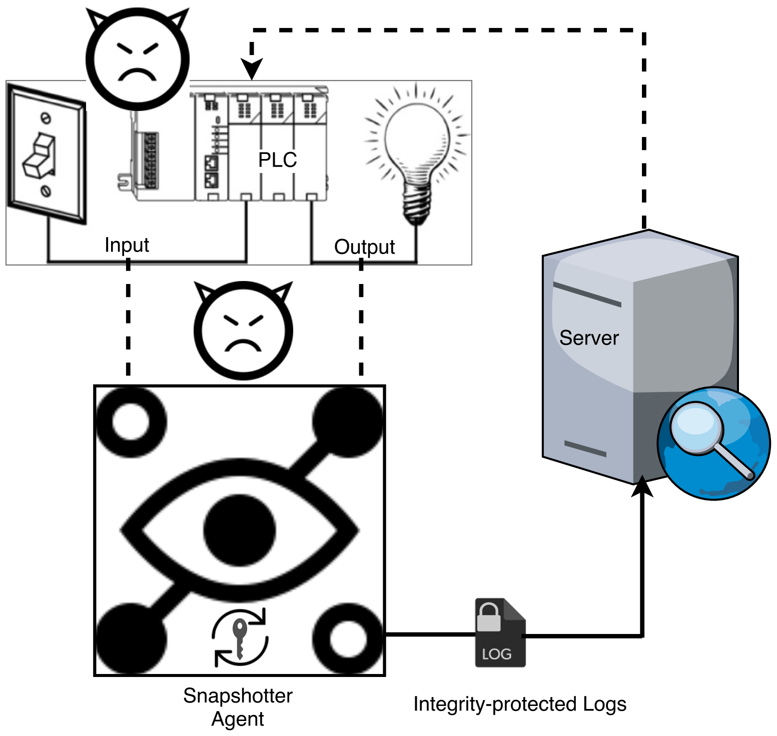


Figure 3.1: Overview of adversarial model and the Snapshotter agent

infected machine to a known clean state (by uploading the legitimate code), activating a backup (redundant) PLC, etc.

Our defense mechanism can be summarized in security-related information gathering and fast forward-secure logging, sending the logs to a trusted server for the purpose of analysis, incident identification and taking effective actions by the server to foil such incidents.

Organization. The background and related works are introduced in Section 3.2. We explain our solution in the context of PLCs in Section 3.3. More implementation details are given in Section 3.4. We provide a security analysis and performance evaluation of our implementation in Section 3.5. Possible future works are presented in Section 3.6. We conclude the paper in Section 3.7.

3.2 Background and Related Work

A host-based intrusion detection system is the common tool used for monitoring specific activities and characteristics of a single device by means of software or appliance-based components known as agents [166]. The agents observe and monitor critical system resources and activities and generate security-related logs which later are used for intrusion detection purposes. However, the logs and alarms generated by a HIDS should be protected against tampering and modification in case the system itself is already compromised. In this regard, a HIDS with the ability to survive such extreme cases seems to be the perfect candidate for our purposes in this study, i.e, securing the ICS components such as PLCs.

Forward-secure logging schemes can provide forward-secure integrity protection of the logs in the post-compromise time. For example, the paper in [198] presents a secure and aggregate logging scheme called Blind-Aggregate-Forward (BAF) along the same lines of [138], without any reliance on online trusted third parties or secure hardware in addition to low computational, storage and communication costs. [168] describes a computationally cheap technique to make pre-compromise era logs unreadable for the adversary or not being modifiable or destroyable in an undetectable manner.

Amongst available works, we found [29] more relevant and applicable. It introduces an interesting tool called PillarBox for the purpose of securely relaying security analytics sources (SAS) data to combat an adversary taking advantage of an advanced malware who can undetectably suppress or tamper with SAS messages in order to hide attack evidence and disrupt intrusion detection. Pillarbox provides two unique features:

- Integrity: by securing SAS data against tampering, even when such data is buffered on a compromised host.
- Stealthiness: by concealing alert-generation patterns and rules on a compromised host, therefore, preventing any leakage of information regarding log mechanism from an adversary who sniffs network traffic before compromise and learns all host state after compromise.

3.3 Proposed Defense Scheme

3.3.1 Adversarial Model

In order to model the capability of adversaries in practice, we consider the extreme case and assume that the adversaries can somehow get access to the PLCs and upload malicious logics, either by physical access or remote access. For example, studies on wind farm security shows that it is very easy to physically get into one wind turbine in the field and have access to its PLC [178]. Also, Stuxnet and Industroyer represent the malware that only requires remote access to take over an industrial control system [122, 126, 128].

Moreover, as the trusted computing base, we assume the monitoring server in our intrusion detection system to be trusted, and all the program running on it is trusted.

3.3.2 Snapshotter Agent

The working flow of our proposed defense mechanism can be summarized in three steps: security related information gathering, incident identification, and mitigation.

Information Gathering. Since we assume a very strong adversarial model who has full access to the device, i.e. PLC and Snapshotter agent including all the digital states and secret keys, we have to have a proper key management scheme to deal with key exposure. This nearly impossible task can be achieved by forward secure techniques, which are well studied in the cryptography community [138, 168, 198, 197]. Counterintuitively, forward security states that in a system that uses unique session keys for different periods or events, the compromise of one session key will not reveal all the keys used *before* this compromise [20]. The most efficient forward secure key management technique is to keep updating the encryption key by a cryptographically secure hash function, meaning that the new key will be updated to the hash value of the current key, and the current key should be removed after that. Thanks to the strong one-wayness of cryptographically secure hash functions, this simple key update trick can achieve forward security [105].

After solving key exposure issue, we need to decide how to protect the integrity of the logs. We can follow the approach introduced in [29]: each log is encrypted by a session key, and the session key is updated in a forward secure way after each encryption. Clearly, using message authentication codes (MAC) is another valid solution, since MAC is designed for protecting the integrity of messages [105]. However, a MAC-based approach requires the Snapshotter agent to send both the plaintext log and its MAC value. It means that the MAC-based approach has two disadvantages. Firstly, the logs are not encrypted, so the content of the logs are exposed to the adversaries, which can possibly improve their knowledge on the system. Secondly, it needs more communication bandwidth, as the MAC value is not inside the log.

We can summarize the control flow of the logging mechanism in one scan cycle on PLCs in Fig. 3.2. The Snapshotter agent keeps checking the occurrence of events/

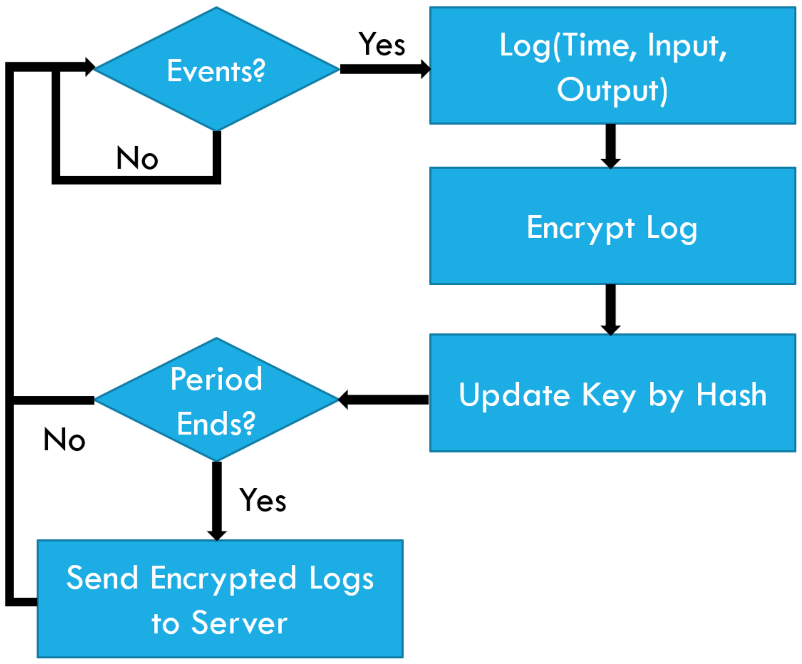


FIGURE 3.2: The control flow of logging mechanism.

state updates of the monitored PLC. The events can simply be value changes at input or output for legacy PLCs, or the events in a modern PLC can contain the flag of overwriting logic and other configuration update, besides the changes at I/O. Once an event happens, one log will be generated, containing the current time, and the current input, output values, possibly with other updating flags in the case of modern PLCs. This log should be encrypted and the key will be overwritten by an updated key equal to the hash value of the current key. Whenever a predefined period ends, all the generated logs in this period will be sent over to the server for incident identification.

This information gathering mechanism achieves the integrity and stealthiness properties we defined in Sec. 3.2, thus it is able to protect the integrity of the logs even when the secret key is exposed, and the adversaries can learn nothing from all the logs, which increase the difficulty in further penetrating the system.

Incident Identification. The server side, upon receiving the encrypted logs from a Snapshotter agent, should first verify the integrity of the logs. Only the validated reports can be used for further incident identification and analysis. The server can use the logged input and time, together with the known PLC logic to simulate the expected output of the PLC[2]. Notice that, a PLC simulator is able to simulate the analog behavior of one PLC as well, and our detection rule should define an acceptable range of fluctuation at an analog output. This acceptable range is essentially one of the hidden parameters in our intrusion detection system, our stealthy logging scheme can prevent adversaries from learning this hidden parameter and exploit this knowledge in the future attacks.

Once the received PLC output does not match the expected (simulated) output, the server will conclude that this PLC has been compromised (the logic has been updated to a malicious one), so the server should take proper actions to recover the industrial process of the system.

Mitigation. If an incident has been identified, one has many different options to mitigate the damage of the attacks. For example, the server can restart the compromised PLCs, and reprogram it to a safe/correct state/logic. If the monitoring server is not able to reset the PLC to a clean state, then a technician must be sent to physically approach the PLC and fix it. In the meanwhile, another uncompromised PLC can be deployed to replace the compromised PLC as a hot backup to continue the industrial process.

[2]Assuming no errors in the controller functionality itself. Note that, even if there is an error in the PLC functionality, our proposed method can capture it as well by using same procedure.

3.3.3 Legacy PLCs vs Modern PLCs

When our solution is implemented in a system using only legacy PLCs, we have to add an additional hardware (e.g. a microcontroller) as the Snapshotter agent to monitor the input/output behavior of legacy PLCs. In contrast, when our logging mechanism is implemented in modern PLCs, we can simply reprogram the firmware of the PLCs, and there will be no additional hardware. More importantly, merging Snapshotter agent with a PLC can facilitate information gathering in the sense of gathering more accurate running status of the PLC.

However, in terms of isolation, the deployment on legacy PLCs has an advantage over modern PLC system, as the PLCs and Snapshotter agents are physically isolated from one another. An adversary, who compromises the legacy PLCs, cannot immediately compromise the logging mechanism, which leads to an easy detection for the server. Snapshotter agent for modern PLCs is running on the same device where the PLC logic is evaluated. Compromising PLCs can easily lead to a compromise of our logging scheme and the encryption keys. Fortunately, due to forward secure key management our solution can maintain security even when the encryption key is exposed to the adversary.

3.4 Implementation Details

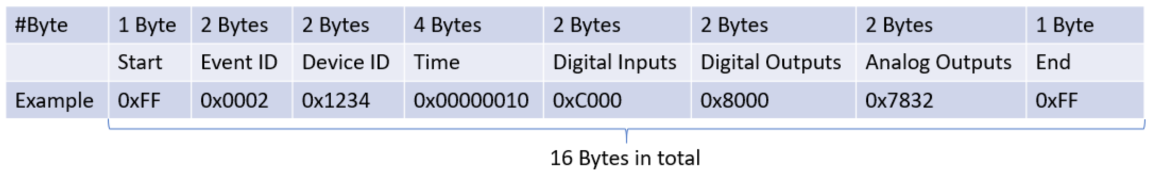
We implemented our Snapshotter agent in the OpenPLC framework on a Raspberry Pi [10]. In order to show the effectiveness of our scheme on modern PLCs and legacy PLCs, we only log the input, output changes in the PLC for checking the integrity of the PLC logic. Thus in the OpenPLC framework, *Read_Input* function and *Write_Output* function have been modified to return 1 if any value changes in the current scan cycle. A single event is encrypted and its key is updated with its hash value such that the old key for encrypting this event will be overwritten right after the encryption. In particular, we use AES-128 [161] and SHA-256[179] as the encryption algorithm and hash function, respectively. The updated key is the first 16 bytes of the hash value of the previous key.

Since the input or output of PLC does not change very often, by only reporting changes of input/output values in the log we can significantly reduce the size of the log. For an application that requires the inputs/outputs of PLCs to be updated very frequently, we suggest to reduce the reporting period or ignore some small fluctuations in the analog output.

After a predefined reporting period has elapsed, the PLC needs to send all the buffered encrypted logs to the server and empty its buffer for the next period. The reporting period is set to 10 seconds in the prototype implementation, but it can be easily adapted to a different value as required. In case no input/output changes within the current period, it still needs to send a log to the server to indicate that it is still online.

To minimize the size of the log we sent, we design one data format to record one event (input or output change) in less than 128 bits, which can fit in one encryption block of AES. The data format is depicted in Fig. 3.3. The first byte is used as an indicator of the start of one event; we set it as 0xFF. The second and third bytes are used to store the event ID in the current time period. Since the implemented PLC is running on 100Hz, and we set the reporting period to be 10 seconds, the maximum number of events can happen in one period is 1000[3]. Therefore, two bytes

[3]This implies that the maximum buffer size for storing logs is 16KB in the current setting, which



#Byte	1 Byte	2 Bytes	2 Bytes	4 Bytes	2 Bytes	2 Bytes	2 Bytes	1 Byte
	Start	Event ID	Device ID	Time	Digital Inputs	Digital Outputs	Analog Outputs	End
Example	0xFF	0x0002	0x1234	0x00000010	0xC000	0x8000	0x7832	0xFF

$\underbrace{\qquad\qquad\qquad\qquad\qquad\qquad}_{\text{16 Bytes in total}}$

FIGURE 3.3: Data format of one event in the log.

are more than enough to represent this maximum number. The fourth and fifth bytes are reserved for the device ID. In total, 65536 devices are allowed to be managed by one server. The following 4 bytes are used to store the time stamp, because in the OpenPLC framework, this is a 4 byte variable. The next six bytes are divided for storing digital inputs, digital outputs and analog output value respectively. Each of them takes two bytes. Since the number of digital inputs and digital outputs on a Raspberry Pi is 14 and 11. Two bytes are enough to store all the input/output pins. Also, only one analog pin can be used on a Raspberry Pi, and this value can be represented by a 16-bit value stored in 2 bytes. At the end, another 0xFF byte is appended to indicate the end of one event log.

Our server is also implemented on the same Raspberry Pi. It waits for the incoming packet from the Snapshotter agent. If the packet is not received on time, the server concludes that the PLC is compromised, we will restart the PLC[4]. If the server gets the packet on time, then we need to use the associated key to decode this packet and update the key stored at the server. After that, we need to check whether the decrypted data has the correct data format or not. If not, then it means the integrity of the packet has been compromised, or some packets are dropped by the adversary

is smaller than the memory size of a typical PLC (e.g. 96KB [173, 83]).

[4]In case of poor network connection with high packet loss rate, we can implement a command which allows the server to ask the Snapshotter agent to resend the logs for last period. This mechanism will reduce the false positive rate in such networked system. Note that this false positive rate is due to the poor network connection; our system does not incur any false positive.

intentionally, so the server does not have a synchronized key with Snapshotter agent. If the integrity is also valid, then we need to extract the input change in this period with its time stamp. This input change and time information is applied to a PLC logic simulator, which will generate the expected output change at the right time. Then this expected output is compared with the output received from the PLCs. If they do not match, then the server knows that the logic running on the PLC has been maliciously modified. Therefore, the server will send a command to restart the PLC and reprogram it with the correct logic in this case.

3.5 Evaluation

In this section, we will evaluate our solution and implementation in terms of security and performance.

3.5.1 Security Analysis

Our adversarial model assumes that the adversary has the capability to upload malicious logic to the PLC that will generate erroneous outputs or further compromise the entire PLC. After the entire PLC is compromised, we assume the attacker can even get access to the encryption key, which will be used to encrypt the next event, together with the encrypted logs that have not been sent to the server yet.

Most importantly, we assume that the attack can not compromise the entire PLC and Snapshotter agent immediately. The logging mechanism will be working properly until a certain point after the compromise initiates. In other words, we assume there exist a critical time window that the attacker has initiated the intrusion, but the

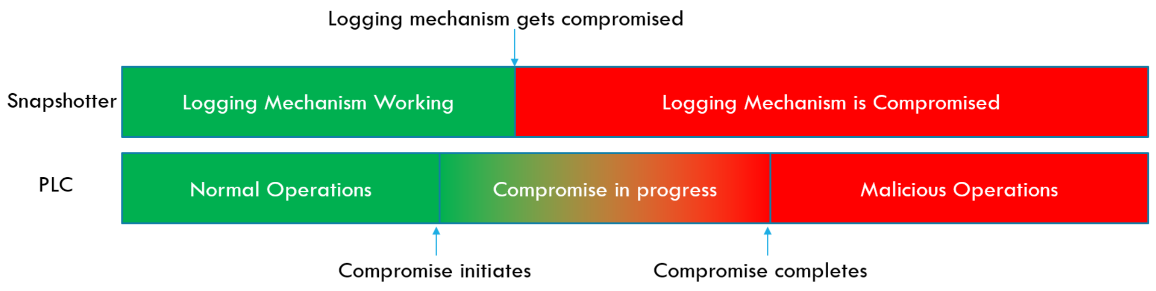


FIGURE 3.4: Time line of one compromise. The critical time window is between the start of compromise and the moment when the logging is compromised.

logging is still working properly. This assumption is used in [29] as well. Essentially the logs generated in this time window tell the server that an intrusion is happening. The overall timeline can be illustrated in Fig. 3.4. The attacker has four different types of actions, and all of them will lead to detection. We list them below:

- **Do nothing.** If the adversary allows the Snapshotter agent to send the logs, the logs containing the information about the intrusion which will lead to detection.

- **Try to decrypt the logs.** Notice that, after compromise the attacker knows the current encryption key. However, the previous logs in the current period have already been encrypted and buffered, and the encryption key is updated in a forward secure way. The attacker has no chance to inverse the hash function in order to get to know the previous encryption key. Therefore, the attacker is not able to decrypt the log.

- **Tamper with the encrypted logs.** The attacker can either send something random or replay a previous log that he/she knows that does not contain any intrusion event information. Since the encryption key is updated after each event, the server with an updated key will not be able to decrypt this tampered log. Therefore, the server will conclude that an intrusion is captured.

- **Packet Dropping.** If the communication between the compromised PLC and the server is blocked, no log will arrive at the server on time. Since the PLC is required to send the report periodically, any PLC which fails to send the log on time will be considered as compromised.

To summarize, there is no way for an attacker to avoid detection by our intrusion detection scheme within our strong adversarial model.

3.5.2 Critical Time Window

As we have explained above, the critical time window describes the only vulnerable time of our system. If an adversary happens to complete his/her attack in this time window, we will not be able to catch this intrusion. Preferably, we want to minimize this time window and understand how long this time window is in a real system. In our prototype implementation, we measured the time for generating logs, encrypting logs and updating the secret key as the length of critical time window. This critical time window is 54 μs per scan cycle (10,000 μs given 100 Hz scan frequency). This implies that if adversaries start an attack at a random time, they will be caught with probability 99.46%.

3.5.3 Detection Latency

Since our solution requires the logs from PLCs to be sent periodically, the largest possible detection latency is a reporting period. Hence, the reporting period should be selected carefully in practice with both security and performance in mind.

3.5.4 Performance Overhead

The program added into the system only contains log generation, log encryption and key update. Thus, according to our experiments, in each scan cycle the additional execution time is *at most* 54 μs, because only those scan cycles which generate events will have additional overhead. Notice that, usually not all of the scan cycle has input/output changes, so our performance overhead can be amortized over multiple scan cycles. Moreover, some of the high end PLCs provide multiple cores [17]. If the Snapshotter agent is running on a different core other than where the PLC logic is running, this can eliminate the performance overhead completely.

3.5.5 Case Study of Attacks

The functionality of the helloworld logic of OpenPLC platform is to keep the connected LED on for two seconds after a falling edge is detected at the switch [9]. We maliciously modified the helloworld logic to keep the LED on for *three* seconds, instead of two seconds, as our malicious logic. After we uploaded the malicious logic to the PLC, and activated the malicious behavior by pushing the switch, our server implementation detects this intrusion successfully.

3.6 Future Directions

Modern PLCs are providing much more flexibility of programming including interrupt service routines, e.g. MicroLogix PLC [7]. This means that we are able to implement

Snapshotter agent in an interrupt service routine, which ought to run at the highest priority in the device, so if attackers cannot prevent the interrupt from being triggered, they will not be able to execute any other codes on the device before Snapshotter agent finishes logging. If we can use the program memory overwritten event to trigger an interrupt, then we do not need to use IO event-based triggering mechanism anymore. This approach can protect the integrity of the code directly, and reduce the size of the packet that is being sent over to the server. Most importantly, this approach pushes the critical time window to the limit, if it is not zero. Unfortunately, the hardware must be adapted to add this interrupt trigger condition, hence no Proof-of-Concept is yet possible. Notice that, even when this trigger condition is implemented, in order to prevent packet dropping attack, we still need to periodically send a liveness signal to the server, which can be easily implemented by a timer-based interrupt.

In order to hide the interrupt service routine from the adversaries, we can apply some code randomization techniques to hide the location of the Snapshotter interrupt service routine [106]. Given these difficulties in exploiting our system, it will significantly demotivate the economically-motivated attackers.

In addition to the new approach, we would like to implement our scheme on a complete industrial control network to test its scalability in the future. Also, we need to investigate a more efficient mitigation strategy, that affects the real industrial process as little as possible.

3.7 Conclusion

In this work, we implemented a logging mechanism suitable for industrial control systems with either legacy or modern devices. This logging mechanism can survive in the extremely strong adversarial model, namely the adversaries can take control of the devices completely, including the secret keys. We also show that our intrusion detection system can work in practice and the performance overhead is extremely small, so it will not hurt the real-time requirement of industrial controllers.

Chapter 4

HMAKE: Legacy-Compliant Multi-factor Authenticated Key Exchange from Historical Data

4.1 Introduction

Cyber-Physical Systems (CPSs), like water treatment systems and nuclear plants, are critical for the daily life of millions of people. However, the security of this kind of systems is always an afterthought, which opens a tremendous attacking surface on CPSs for malicious adversaries [65]. Even worse, many legacy devices with very limited or no security protection are still in use. Since they have been running for decades, it becomes a non-trivial task to upgrade or replace them. Therefore, security enhancements of legacy devices are highly demanded in practice now. As the first step towards a secure system, we need to protect the communication between the devices in the field and the servers/control centers, because most of the devices are required to report their status and data acquired in the field to the server, and they accept commands from the server. In the context of CPS, this kind of servers is usually called supervisory control and data acquisition (SCADA) systems.

In the existing literature, many end-to-end encryption and message authentication methods are suggested between controllers and SCADA system [35, 95], but none of them answered the question about how to establish such a secure communication channel. Of course, one can simply use a single factor authenticated key exchange protocol [48, 64], but can we enhance its security by introducing another authentication factor? Because it is a machine-to-machine (M2M) authentication, the existing two/multi-factor authenticated key exchange (AKE) protocols [49, 202, 37, 90, 82], which usually use passwords or fingerprints as the second factor, do not apply here. Multi-factor M2M AKE might be instantiated from the generic framework [69] by Fleischhacker et al., which allows one to build a protocol by securely mixing multiple types and quantities of authentication factors such as low-entropy (one-time) passwords/PINs, high-entropy private/public keys and biometric factors. However, their framework does not cover the authentication factors that are lightweight while being able to satisfy leakage resilience. We have to find another authentication factor on the server, and it should have a stronger security level from a conventional secret key stored on the same machine.

Recall that CPS devices keep sending data to the SCADA system for monitoring. Actually, for future data analysis, the historical data in most of the SCADA systems is stored in a dedicated process historian, instead of their main servers [174]. This directly implies that the historical data has a different security level from secret keys. Moreover, a secret key, usually hundreds of bits, can be leaked very fast in a security breach, but a large database on the same server will clearly at least slow down the secret leakage by a few orders of magnitude, and consequently implies a different security level. Therefore, a secret key, a database of historical data stored in a historian, and a database of data associated tags stored on a SCADA server are the perfect authentication factors with three different security levels, such that compromising one factor does not lead to a corruption of another authentication factor. As another fact, the historical data and its tag are growing all the time, so a piece of historical data leaked in the past may not be valid as an authentication factor soon after. This makes an impersonation even harder.

Existing Historical Data based Authentication Protocols. The usage of historical data as an authentication factor in an authentication protocol was introduced in [38] and further developed in [39] at ESORICS'16. The early scheme [38] uses the historic data straightforwardly as a symmetric key shared between the client and the server. This imposes a non-trivial storage overhead to the client, which is sometimes infeasible for a resource-constrained CPS device. Recently Chan et al. [39] introduced a scalable historical data based two-factor authentication scheme (which will be referred to as CWZT scheme). Namely, the first authentication factor is a long-term symmetric key and the second authentication factor is a dynamically growing set of secret tags associated with historical data. The CWZT protocol is wisely derived from the proof of retrievability (PoR) protocol [170], in which the server authenticates itself to the client by proving that it possesses all historical data sent by the client. As one of their major contributions, the CWZT protocol only requires the client to store a small constant-sized secrets (e.g., 512 bits), which well fits CPS devices. Chan et al. also introduced historical tag leakage resilience in a bounded-storage model [14, 58] as its security feature, so that partial historical tag leakage does not affect much of its security.

Vulnerabilities of the CWZT Protocols. (1) According to our analysis in Section 4.4, the CWZT protocol is vulnerable to a tag stealing attack. In short, we show that an adversary can steal *all* the historical tags through legitimate interactions with the server, given only *one piece of historical tag* (associated with one data piece) that is somehow leaked. Notice that the *partial historical tag leakage* is allowed in the adversarial model of the CWZT protocol, and was claimed as one of the major contributions in [39]. (2) In [39], the authors suggested to use the first authentication factor to protect the transmission of the second authentication factor (tags). This completely deviates from the motivation of having two authentication factors. Thus how to secure the transmission of data and tags from the client to the server is still an open problem.

Our Contributions. Due to the vulnerabilities and limitations of the existing protocols mentioned above, we cannot simply extend the existing authentication protocols to an AKE protocol. We have to reconsider the fundamental authentication problem based on historical data, and redesign a new AKE protocol from scratch. More specifically, we made four significant contributions as follows:

1. We analyze the stat-of-the-art historical data based authentication protocol (the CWZT protocol [39] proposed at ESORICS'16) and propose a tag-stealing attack which breaks the security claim of the CWZT protocol via legitimate interactions.

2. To build a solid theoretical foundation of our proposed HMAKE protocols, and to avoid repeating the mistakes in the previous designs, we are the first to formally define two indistinguishability-based security models for HMAKE,

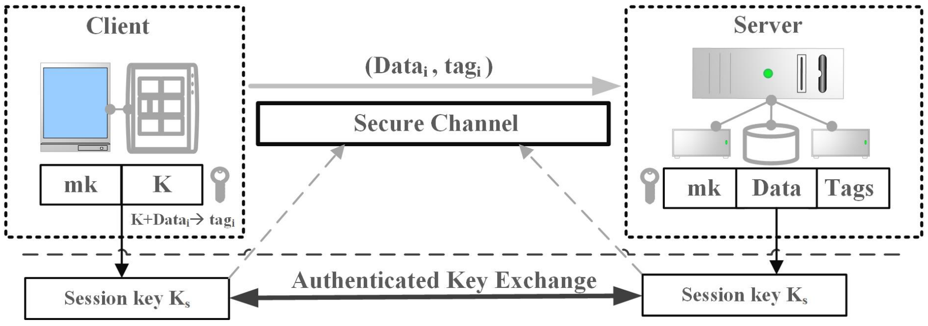


FIGURE 4.1: Overview of Our HMAKE Protocol

and later we analyze our proposed protocols in these security models.

3. As one of the main contributions of this paper, we introduce two HMAKE protocols Π_{woFS} (without forward secrecy) and Π_{FS} (with forward secrecy) and proved their security in the random oracle model.

4. To show the impact of our protocols in the real world, we demonstrate how to deploy our protocols in the field to enhance legacy devices. Also, we implemented Π_{woFS} and Π_{FS}, and evaluated their performance experimentally.

Technical Overview. An overview of our first HMAKE protocol Π_{woFS} is presented in Fig. 4.1. The client device and the server share a master key (mk) as their first authentication factor. When the client sends data to the server, it would generate a secret tag associated with the data using a tag generation key K. The server stores all tuples $\{(Data_i, tag_i)\}$ separately as its second and third authentication factors respectively, while the client only needs to store K as its second authentication factor. The client only has two authentication factors due to its limited storage space, i.e. not storing the historical data. In our HMAKE protocols, both parties can use their authentication credentials to run the key exchange procedure to generate a session key to establish a secure channel, that is used to protect the underlying data and tag transmission.

In addition, Π_{woFS} still has a remarkable security feature called historical tag leakage resilience, such that a small portion of tag leakage will not affect the security of Π_{woFS} much. Notice that although this feature was first introduced in [39], they failed to achieve it due to the tag stealing attack we introduced. Also, because of the clear separation of the two authentication factors in Π_{woFS}, Π_{woFS} can be easily used to enhance the security of legacy CPS devices. An additional device with the second factor can be attached to a legacy device (with the first factor embedded), intercept its traffic, and complete most of the computation in Π_{woFS}.

One limitation of Π_{woFS} is that it can only defend against *static* bounded-leakage regarding the historical tags, and it does not provide perfect forward secrecy. In a static bounded-leakage model, the adversary can only learn a fraction of the secret tags *at the beginning of the security game*. Nevertheless, the static bounded-leakage resilience is still valuable and useful for HMAKE in practice since the leaked tags will be out-dated quickly when the historical data is growing. Theoretically, an attacker may try to adaptively attack many sessions as formulated in the seminal work about entity authentication model [21]. To achieve this adaptive bounded-leakage resilience and perfect forward secrecy, we design the second HMAKE protocol Π_{FS}. In Π_{FS}, we use the first protocol Π_{woFS} as a compiler to transform any passively secure two-message key exchange (TKE) protocols to be an actively secure HMAKE protocol. Because the session key does not depend on the authentication keys (unlike Π_{woFS}), Π_{FS} can resist adaptive bounded-leakage, i.e., the adversary can get access to a bounded number of valid historical tags at any time of the security experiment.

4.2 Related Work

Lightweight AKE Protocols. Due to the limitations of power constrained devices, e.g., sensor networks or IoT devices, researchers have been dedicated to develop lightweight multi-factor AKE protocols in conjunction with specific communication models or application scenarios. For example, the lightweight multi-factor AKE protocols proposed in [49, 82] are designed for wireless sensor networks (WSN), and there are many protocols [37, 188] for Internet-of-Things (IoT). Recently, Dua et al. [55] proposed a protocol to protect the communication of vehicles in smart cities. In [41], Chattaraj et al. proposed an AKE protocol for cloud computing services. For different application scenarios and computation power of players, different authentication factors might be involved. The commonly used authentication factors would be the long-term symmetric key and users' memorable password. Most of the long-term symmetric key based lightweight AKE schemes, e.g.,[49, 37, 82, 194], require some tamper-proof devices (such as smart cards) to store the authentication key. To enhance its security, a protocol might also incorporate biometric factors [49, 37] into authentication that has more entropy than a password. However, none of the above lightweight AKE protocol covers the leakage resilient property as our proposals.

Cryptographic Primitives for PFS. Considering the importance of PFS, many AKE schemes are proposed with PFS based on Diffie-Hellman key exchange (DHKE), e.g., [49, 37, 193, 192, 82]. Fortunately, some results (e.g., [92, 44]) have shown that the DHKE protocol might be feasible to be realized with the elliptic curves cryptography (ECC) optimized for embedded systems. We also instantiate our protocol Π_{FS} with ECC based DHKE protocol for comparison.

Generic AKE Compilers. A research line related to our second protocol Π_{FS} is the AKE compiler that is to securely combine authentication protocols (AP) with passively secure key exchange protocols (KE) in a modular and generic manner, e.g., [88, 130, 69, 195]. However, no existing AKE compilers leverage historical data based authentication protocol as a building block. Our protocol Π_{FS} presents a new way to realize AKE compilers.

4.3 Preliminaries

General Notations. Let $\kappa \in \mathbb{N}$ denote the security parameter and 1^κ be a string of κ ones. We let $[n] = \{1, \ldots, n\} \subset \mathbb{N}$ denote the set of integers between 1 and n. We write $a \overset{\$}{\leftarrow} S$ to denote the operation sampling a uniform random element from a set S. We let $\|$ denote the concatenation (operation) of two strings.

Random Oracles. Bellare and Rogaway [22] first used the random oracle as a tool to prove the security of cryptographic schemes. In this paper, we assume that the hash function $h(\cdot)$ is modeled as a random oracle. A random oracle is stateful. Namely, on input a value $m \in \{0, 1\}^*$, the random oracle query $h(m)$ proceeds as follows: (i) With respect to the first query on m, the oracle just returns a true random value r_m from the output space, and records the tuple (m, r_m) into its query list HL; (ii) If $m \in \mathsf{HL}$, then the oracle returns its associated random value r_m recorded in HL.

As in [51], we use a uniformly random salt $\chi \overset{\$}{\leftarrow} \mathcal{X}$ as input of h to sample a random oracle $h(\chi, \cdot)$, where $\mathcal{X}$ is a salt space. When the salt is clear in the context, we may write $h(\cdot)$ instead of $h(\chi, \cdot)$ for simplicity. The random salt can be used to prevent vulnerabilities introduced in [51].

Passively Secure Two-message Key Exchange. We consider a two-message key exchange (TKE) protocol in which the session key is established within only two protocol passes. In each protocol pass, a single message is sent by a party. We further assume that each player of the protocol does not hold any long-term secret key for simplicity. Specifically, a general TKE protocol may consist of three polynomial time algorithms $(\mathsf{TKE.Setup}, \mathsf{TKE.MSG}, \mathsf{TKE.SKG})$ which are defined as follows:

- $pms \leftarrow \mathsf{TKE.Setup}(1^\kappa)$: On input 1^κ, the setup algorithm outputs pms, a set of system parameters. We assume the other algorithms may implicitly use pms.

- $m_{\mathsf{id}_1} \xleftarrow{\$} \mathsf{TKE.MSG}(\mathsf{id}_1, r_{\mathsf{id}_1}, m_{\mathsf{id}_2})$: The message generation algorithm takes as input a party's identity id_1, a randomness $r_{\mathsf{id}_1} \xleftarrow{\$} \mathcal{R}_{\mathsf{TKE}}$ and a message $m_{\mathsf{id}_2} \in \mathcal{M}_{\mathsf{TKE}}$ received from party id_2, and outputs a message $m_{\mathsf{id}_1} \in \mathcal{M}_{\mathsf{TKE}}$ to be sent, where $\mathcal{R}_{\mathsf{TKE}}$ is the randomness space and $\mathcal{M}_{\mathsf{TKE}}$ is the message space. Note that if id_1 is the sender then $m_{\mathsf{id}_2} = \emptyset$.

- $K \leftarrow \mathsf{TKE.SKG}(\mathsf{id}_1, r_{\mathsf{id}_1}, \mathsf{id}_2, m_{\mathsf{id}_2})$: The session key generation algorithm takes as an input the participants' identities id_1 and id_2, the randomness r_{id_1} and the received message m_{id_2} from party id_2, and outputs a session key $K \in \mathcal{K}_{\mathsf{TKE}}$, where $\mathcal{K}_{\mathsf{TKE}}$ is the session key space.

We say that the $\mathsf{TKE.SKG}$ algorithm is correct, if for all random values $r_{\mathsf{id}_1}, r_{\mathsf{id}_2} \xleftarrow{\$} \mathcal{R}_{\mathsf{TKE}}$ and all messages $m_{\mathsf{id}_1} \xleftarrow{\$} \mathsf{TKE.MSG}(\mathsf{id}_1, r_{\mathsf{id}_1}, \emptyset)$ and $m_{\mathsf{id}_2} \xleftarrow{\$} \mathsf{TKE.MSG}(\mathsf{id}_2, r_{\mathsf{id}_2}, m_{\mathsf{id}_1})$, it holds that $\mathsf{TKE.SKG}(\mathsf{id}_1, r_{\mathsf{id}_1}, \mathsf{id}_2,$
$m_{\mathsf{id}_2}) = \mathsf{TKE.SKG}(\mathsf{id}_2, r_{\mathsf{id}_2}, \mathsf{id}_1, m_{\mathsf{id}_1})$

We define a security experiment for passively secure TKE protocols as follows.

Security Experiment: The security experiment is carried out as a game between a challenger $\mathcal{C}$ and an adversary $\mathcal{A}$ based on a protocol TKE. During the setup phase,

$\mathcal{C}$ generates the parameters $pms \leftarrow \mathsf{TKE.Setup}(1^{\kappa})$ and two identities $\{\mathcal{ID}_1, \mathcal{ID}_2\}$ of protocol participants. The adversary is given pms and all identities as input. Next, $\mathcal{A}$ will interact with $\mathcal{C}$ via asking at most $d \in \mathbb{N}$ times $\mathsf{Execute}(\mathsf{id}_1, \mathsf{id}_2)$ query; for each $\mathsf{Execute}$ query, $\mathcal{C}$ runs a fresh protocol instance between id_1 and id_2, and returns the corresponding protocol messages' transcript T and session key K to $\mathcal{A}$. At some point, $\mathcal{A}$ submits a challenge request $\ltimes$. Upon receiving $\ltimes$, $\mathcal{C}$ runs a new protocol instance obtaining the transcript T^* and the session key K_1^*, samples a random key K_0^*, and tosses a fair coin $b \in \{0, 1\}$. Then, $\mathcal{C}$ returns (T^*, K_b^*) to $\mathcal{A}$. After the challenge query, $\mathcal{A}$ may continue making $\mathsf{Execute}(\mathsf{id}_1, \mathsf{id}_2)$ queries. Finally, $\mathcal{A}$ may terminate and output a bit b'.

Definition 1. *We say that a two-message key exchange protocol* TKE *is* $(t, \epsilon_{\mathsf{TKE}})$-passively-secure *if for all probabilistic polynomial time (PPT) adversaries running the above experiment in time t, it holds that*
$|\Pr[b = b'] - 1/2| \leq \epsilon_{\mathsf{TKE}}$.

4.4 Cryptanalysis of the CWZT Scheme

In this section, we revisit the security property of CWZT scheme [39, §5.1] regarding the resilience to the leakage of historical tags. We will introduce an attack to subvert the leakage resilience of CWZT scheme. Note that the leakage resilience is an intrinsic property that distinguishes historical data relevant authentication factors from other symmetric key based authentication factors.

$$
\begin{array}{rcl}
\text{Verifier } \mathsf{id_C} & & \text{Prover } \mathsf{id_S} \\
\hline
 & \text{Initialization} & \\
sk^1_{\mathsf{id_C},\mathsf{id_S}} = mk \xleftarrow{\$} \{0,1\}^\kappa & \xrightarrow{\quad sk^1_{\mathsf{id_C}} \quad} & sk^1_{\mathsf{id_S},\mathsf{id_C}} := sk^1_{\mathsf{id_C},\mathsf{id_S}} \\
K \xleftarrow{\$} \mathbb{Z}_p,\ K' \xleftarrow{\$} \{0,1\}^\kappa & \textsf{secure channel} & \\
L := 0 & & \\
sk^2_{\mathsf{id_C},\mathsf{id_S}} = (K, K') & & sk^2_{\mathsf{id_S},\mathsf{id_C}} = \mathcal{D} = \emptyset \\
\hline
 & \text{Tag Generation: for the } i\text{-th data} & \\
k_i := f(K', i) & \xrightarrow{\quad i, d_i, t_i \quad} & \\
t_i := K \cdot h(d_i) + k_i & & \\
L := L + 1 & & \text{store } (i, d_i, t_i) \to \mathcal{D} \\
\hline
 & \text{Online Authentication} & \\
\text{Sample } z \text{ random indices:} & & \\
\mathsf{I} = (I_1, I_2, \ldots, I_z) \xleftarrow{\$} [L] & & r' := E(sk^1_{\mathsf{id_S},\mathsf{id_C}}, r) \\
r \xleftarrow{\$} \{0,1\}^\kappa & \xrightarrow{\quad \mathsf{I}, r \quad} & \text{for } i \in \mathsf{I}\text{: } (d_i, t_i) \leftarrow \mathcal{D} \\
r' := E(sk^1_{\mathsf{id_C},\mathsf{id_S}}, r) & & X := \sum_{i \in \mathsf{I}} f(r', i) \cdot h(d_i) \\
K_{\mathsf{I}} := \sum_{i \in \mathsf{I}} f(K', i) \cdot f(r', i) & \xleftarrow{\quad X, Y \quad} & Y := \sum_{i \in \mathsf{I}} f(r', i) \cdot t_i \\
Y' := K_{\mathsf{I}} + K \cdot X & & \\
\text{accept iff } Y' = Y & &
\end{array}
$$

FIGURE 4.2: The CWZT Protocol [39].

4.4.1 Protocol Review

We first briefly review the CWZT scheme. Let $\mathbb{Z}_p$ be an abelian group with prime order p that has κ bits. The CWZT protocol makes use of two pseudorandom functions $f : \{0,1\}^\kappa \times \{0,1\}^* \to \mathbb{Z}_p$ and $E : \{0,1\}^\kappa \times \{0,1\}^\kappa \to \{0,1\}^\kappa$, and a cryptographic hash function $h : \{0,1\}^* \to \mathbb{Z}_p$. The protocol running between a verifier $\mathsf{id_C}$ and a prover $\mathsf{id_S}$ is shown in Fig. 4.2.

4.4.2 A Tag Stealing Attack

Here we introduce an attack where an attacker $\mathcal{A}$ who knows one secret tuple $(h(d_j), t_j)$ is able to steal all the other historical tags, i.e., $\{(h(d_i), t_i)\}_{i \in [L], i \neq j}$. In our attack, we exploit the fact that there is no authentication to the verifier. This fact enables an attacker masquerading the verifier $\mathsf{id_C}$ to choose two malicious selection sets I_1 and I_2 which only differ in one index that is associated with the target token which we want to steal. In a nutshell, we need two assumptions that (i) $\mathcal{A}$ has corrupted the first authentication key $sk^1_{\mathsf{id_C},\mathsf{id_S}} = sk^1_{\mathsf{id_S},\mathsf{id_C}}$, and (ii) $\mathcal{A}$ learns one secret tuple $(h(d_j), t_j)$ with arbitrary index j. Note that this is allowed by the CWZT scheme [39].

In the following, we show how the attacker $\mathcal{A}$ steals the i^*-th token (for $i^* \in [L]$ and $i^* \neq j$) holding by prover $\mathsf{id_S}$.

- $\mathcal{A}$ somehow corrupts $sk^1_{\mathsf{id_C},\mathsf{id_S}}$ and (d_j, t_j).
- $\mathcal{A}$ masquerades as the verifier $\mathsf{id_C}$ to choose a randomness r and a selection set I_1, such that $i^* \notin \mathsf{I}_1$ and $j \in \mathsf{I}_1$.
- $\mathcal{A}$ sends (I_1, r) to $\mathsf{id_S}$ in a session, and receives the authentication messages (X_1, Y_1).
- In another session, $\mathcal{A}$ chooses a selection set I_2 by replacing the index j with i^*, and sends (I_2, r) to $\mathsf{id_S}$ in another session, and receives the authentication messages (X_2, Y_2).
- $\mathcal{A}$ computes $r' := E(sk^1_{\mathsf{id_C},\mathsf{id_S}}, r)$, $f(r', j)$, and $f(r', i^*)$.
- Then $\mathcal{A}$ can obtain $h(d_{i^*})$ and t_{i^*} by Equation 4.1 and Equation 4.2 respectively.

$$\begin{aligned} h(d_{i^*}) &= \frac{X_2 - X_1 + f(r', j) \cdot h(d_j)}{f(r', i^*)} \\ &= \frac{\left(\sum_{i \in \mathsf{I}_1 \setminus j} f(r', i) \cdot h(d_i) + f(r', i^*) \cdot h(d_{i^*})\right) - \sum_{i \in \mathsf{I}_1 \setminus j} f(r', i) \cdot h(d_i)}{f(r', i^*)}. \end{aligned} \tag{4.1}$$

$$
\begin{aligned}
t_{i^*} &= \frac{Y_2 - Y_1 + f(r', j) \cdot t_j}{f(r', i^*)} \\
&= \frac{(\sum_{i \in \mathsf{I}_1 \setminus j} f(r', i) \cdot t_i + f(r', i^*) \cdot t_{i^*}) - \sum_{i \in \mathsf{I}_1 \setminus j} f(r', i) \cdot t_i}{f(r', i^*)}.
\end{aligned} \tag{4.2}
$$

By repeating the above attack steps, the attacker can obtain other authentication tokens as it wishes.

Attack Discussion. Note that the computation on the authentication proof Y is a linear combination of the secrets derived from those authentication factors (i.e., ephemeral key $f(r', i)$ generated based on the symmetric key $sk^1_{\mathsf{id_S},\mathsf{id_C}}$ and historical tags t_i). However, the ephemeral keys derived by the first authentication factor $sk^1_{\mathsf{id_S},\mathsf{id_C}}$ cannot provide any protection for the historical tags in the computation of Y, since $sk^1_{\mathsf{id_S},\mathsf{id_C}}$ might be corrupted. Hence, the security of those authentication factors should be considered *independently* in the protocol design. Since the verifier (i.e., the client $\mathsf{id_C}$) cannot be explicitly authenticated (within two passes), the selection set I can be malicious which implies that the the authentication proof Y is generated maliciously as well. Hence, the selection set should be determined by both parties instead. Based on the above observations, we will show how to avoid this problem in our HMAKE constructions.

4.5 HMAKE Security Model

In this section, we define new indistinguishability-based security models for historical data based multi-factor authenticated key exchange protocols (HMAKE). In these security models, we will formulate the security goals that our upcoming HMAKE protocols can achieve. The new models basically follow from the security models for AKE in literature, e.g., [21, 69, 131, 43]. In contrast to previous models, we particularly formulate the authentication factors related to historical data, and the security property regarding leakage resilience.

Execution Environment. Here we consider an environment where two honest parties exist, i.e., an honest client $\mathsf{id_C}^*$ and an honest server $\mathsf{id_S}^*$. In the following, we let $\mathcal{ID}$ be a general identity to denote one of the honest parties in $\{\mathsf{id_C}^*, \mathsf{id_S}^*\}$.[1] However, we would allow an adversary to register new malicious clients. The client $\mathsf{id_C}$ and the server $\mathsf{id_S}$ would share a long-term symmetric authentication key $sk^1_{\mathsf{id_C},\mathsf{id_S}}$ as the first authentication factor. The second authentication key of a client is denoted by $sk^2_{\mathsf{id_C},\mathsf{id_S}}$ (which is used to verify the authentication message from $\mathsf{id_S}$). Besides the first symmetric authentication factor shared with the client, the server $\mathsf{id_S}$ would store distinct authentication factors, i.e., historical data $\mathcal{D}_1$ and the corresponding secret historical tags $\mathcal{D}_2$, where each piece of historical data is associated with a secret historical tag. We denote them by $sk^2_{\mathsf{id_S},\mathsf{id_C}} = \mathcal{D}_1$ and $sk^3_{\mathsf{id_S},\mathsf{id_C}} = \mathcal{D}_2$ such that $sk^\alpha_{\mathsf{id_S},\mathsf{id_C}} = (sk^\alpha_{\mathsf{id_S},\mathsf{id_C}}(1), sk^\alpha_{\mathsf{id_S},\mathsf{id_C}}(2), \ldots, sk^\alpha_{\mathsf{id_S},\mathsf{id_C}}(L))$ for $\alpha \in \{2, 3\}$ that comprises of the sub-authentication keys denoted by $sk^\alpha_{\mathsf{id_S},\mathsf{id_C}}(i)$ for $i \in [L]$, where $L \in \mathbb{N}$ is the number of the stored historical data. Moreover, each party also maintains states $\{cst^i\}$ denoting the i-th authentication factor corruption status $cst^i \in \{\mathsf{exposed}, \mathsf{fresh}\}$ for $i \in \{1, 2, 3\}$. For example, if $sk^2_{\mathsf{id_S},\mathsf{id_C}}$ is corrupted, the party $\mathsf{id_S}$ must have $cst^i_{\mathsf{id_S},\mathsf{id_C}} =$ $\mathsf{exposed}$. We assume the authentication factors of a party are stored independently, so that the corruption of a factor does not affect the others. To emulate the protocol

[1]Here we only consider two honest parties for simplicity. Multiple honest parties' security can be asymptotically derived from the two-party case.

executions, we assume that each party $\mathcal{ID}$ can carry out at most $\rho \in \mathbb{N}$ sessions that are modeled by a set of oracles $\{\pi^u_{\mathcal{ID}} : i \in [\ell], u \in [\rho]\}$. All oracles can have access to the authentication keys of its owner. Moreover, we assume each oracle $\pi^u_{\mathcal{ID}}$ maintains a list of independent internal state variables: (i) $\Phi^u_{\mathcal{ID}}$ – session decision $\Phi^u_{\mathcal{ID}} \in \{\texttt{accept}, \texttt{reject}\}$; (ii) $\mathsf{pid}^u_{\mathcal{ID}}$ – identity of the intended communication partner; (iii) $K^u_{\mathcal{ID}}$ – session key of $\pi^u_{\mathcal{ID}}$; (iv) $T^u_{\mathcal{ID}}$ – protocol messages orderly sent and received by $\pi^u_{\mathcal{ID}}$. We assume that the session key $K^u_{\mathcal{ID}}$ will be assigned with a non-empty value if and only if $\Phi^u_{\mathcal{ID}} = \texttt{accept}$. [2]

Adversarial Model. To model the power of an active adversary $\mathcal{A}$, we realize $\mathcal{A}$ as a probabilistic polynomial time (PPT) algorithm that can ask the following queries:

- $\mathsf{Send}(\mathcal{ID}, u, m)$: The adversary can send any message m to the oracle $\pi^u_{\mathcal{ID}}$ via this query. Oracle $\pi^u_{\mathcal{ID}}$ will respond the next protocol message m^* (if any) to be sent according to the protocol specification and its internal states. An oracle of the honest client $\mathsf{id_C}^*$ is initiated via sending the oracle the first message $m = \top$ consisting of a special initialization symbol $\top$. The oracle variables will be updated accordingly (following the protocol specification) after each Send query.

- $\mathsf{RevealKey}(\mathcal{ID}, u)$: The oracle $\pi^u_{\mathcal{ID}}$ responds with the contents of $K^u_{\mathcal{ID}}$.

- $\mathsf{Corrupt1}(\mathcal{ID}_1, \mathcal{ID}_2)$: For honest parties $(\mathcal{ID}_1, \mathcal{ID}_2) \in \{\mathsf{id_C}^*, \mathsf{id_S}^*\}$, this query returns the first authentication key $sk^1_{\mathcal{ID}_1,\mathcal{ID}_2}$ of an honest party $\mathcal{ID}_1$, and sets $cst^1_{\mathcal{ID}_1,\mathcal{ID}_2} = cst^1_{\mathcal{ID}_2,\mathcal{ID}_1} := \mathsf{exposed}$.

[2] Note that, throughout the paper, the superscript u of an oracle or a state of an oracle is the index of the oracle, while the other superscripts are 1, 2 or 3 (e.g. $sk^1_{\mathsf{id_C},\mathsf{id_S}}$, $sk^2_{\mathsf{id_C},\mathsf{id_S}}$, and $sk^3_{\mathsf{id_C},\mathsf{id_S}}$) denoting which authentication factor it is referring to. The subscript always represents the ID of a user.

- $\mathsf{Corrupt2}(\mathcal{ID}_1, \mathcal{ID}_2)$: For honest parties $(\mathcal{ID}_1, \mathcal{ID}_2) \in \{\mathsf{id_C}^*, \mathsf{id_S}^*\}$, this query returns the second authentication key $sk^2_{\mathcal{ID}_1,\mathcal{ID}_2}$ of an honest party $\mathcal{ID}_1$, and sets $cst^2_{\mathcal{ID}_1,\mathcal{ID}_2} := \mathsf{exposed}$.

- $\mathsf{Corrupt3}$: This query returns the third authentication key $sk^3_{\mathsf{id_S}^*,\mathsf{id_C}^*}$, and sets $cst^3_{\mathsf{id_S}^*,\mathsf{id_C}^*} := \mathsf{exposed}$.

- $\mathsf{RevealR}(\mathcal{ID}, u)$: This query returns the randomness generated by $\pi^u_{\mathcal{ID}}$.

- $\mathsf{HTLeak}(i)$: This query returns the i-th sub-key $sk^3_{\mathsf{id_S}^*,\mathsf{id_C}^*}(i)$.

- $\mathsf{RegClient}(\mathsf{id}_{\mathsf{C}i}, sk^1_{\mathsf{id}_{\mathsf{C}i},\mathsf{id_S}^*}, sk^2_{\mathsf{id}_{\mathsf{C}i},\mathsf{id_S}^*}, sk^2_{\mathsf{id_S}^*,\mathsf{id}_{\mathsf{C}i}}, sk^3_{\mathsf{id_S}^*,\mathsf{id}_{\mathsf{C}i}})$: This query allows the adversary to register malicious clients and authentication keys. If $\mathsf{id}_{\mathsf{C}i}$ exists, then the old keys will be replaced with the input ones.

- $\mathsf{Test}(\mathcal{ID}, u)$: If the oracle $\pi^u_{\mathcal{ID}}$ has $\Phi^u_{\mathcal{ID}} \neq \mathtt{accept}$, then the oracle returns a failure symbol $\perp$. Otherwise, it flips a random bit b, samples a random key K_0, and sets $K_1 = K^u_{\mathcal{ID}}$. Finally, the key K_b is returned. We call the oracle $\pi^u_{\mathcal{ID}}$ in this query as a *test oracle.*

Secure HMAKE Protocols. We first review a notion called *matching conversations* that was first introduced in [21] to formulate the relation between two sessions. We will use a variant that is refined in [89].

Matching Conversations. An oracle $\pi^u_{\mathcal{ID}}$ is said to have a matching conversation to an oracle $\pi^v_{\mathsf{pid}^u_{\mathcal{ID}}}$, if either (i) $\pi^u_{\mathcal{ID}}$ has sent all protocol messages and $T^v_{\mathsf{pid}^u_{\mathcal{ID}}}$ is a prefix of $T^u_{\mathcal{ID}}$, or (ii) $\pi^v_{\mathsf{pid}^u_{\mathcal{ID}}}$ has sent all protocol messages and $T^u_{\mathcal{ID}}$ is a prefix of $T^v_{\mathsf{pid}^u_{\mathcal{ID}}}$. We also call $\pi^v_{\mathsf{pid}^u_{\mathcal{ID}}}$ meeting all above conditions to be the partner oracle of $\pi^u_{\mathcal{ID}}$.

Correctness. We say a HMAKE protocol Π is correct, if two accepted oracles $\pi^u_{\mathsf{id_C}^*}$ and $\pi^v_{\mathsf{id_S}^*}$ have matching conversations, then both oracles should generate the same session key.

We will use the variable $\mathsf{MN} \in \{\mathsf{FS}, \mathsf{woFS}\}$ to denote the HMAKE security either with PFS (Perfect Forward Secrecy) or without PFS (woFS). In the following, we present a unified security experiment with/without FS based on MN. For a HMAKE protocol without PFS, we only define static historical tag leakage, explicit authentication for the server, and the implicit authentication for the client. However, for a HMAKE protocol with PFS, we define mutual explicit authentication and adaptive historical tag leakage.

HMAKE Security Experiment (Π, MN): A challenger $\mathcal{C}$ will play a game with an adversary $\mathcal{A}$ based on a target HMAKE protocol Π and the security variable MN. In the initialization phase of the game, $\mathcal{C}$ first implements a collection of oracles $\{\pi^u_{\mathcal{ID}} : \mathcal{ID} \in \{\mathsf{id_C}^*, \mathsf{id_S}^*\}, u \in [\rho]\}$ for the honest client $\mathsf{id_C}^*$ and the honest server $\mathsf{id_S}^*$ respectively. All authentication keys are generated according to the protocol specifications. $\mathcal{C}$ gives the adversary $\mathcal{A}$ all identities as input. There are two phases in the game, and in each phase, distinct queries can be asked. In the first phase, $\mathcal{A}$ is allowed to ask queries to HTLeak, to model static historical tag leakage. $\mathcal{A}$ can send $\mathcal{C}$ a symbol $\vdash$ to switch to the next phase. In the second phase, $\mathcal{A}$ can ask a polynomial number of queries to Send, $\mathsf{Corrupt1}$, $\mathsf{Corrupt2}$, $\mathsf{Corrupt3}$, $\mathsf{RevealKey}$, $\mathsf{RevealR}$, and $\mathsf{RegClient}$. If $\mathsf{MN} = \mathsf{woFS}$, the HTLeak query is not allowed in the second phase. However, if $\mathsf{MN} = \mathsf{FS}$, the adversary can query HTLeak in this phase to model adaptive leakage. During the second phase, $\mathcal{A}$ may issue a $\mathsf{Test}(\mathcal{ID}, u)$ query at most once. After the Test query, $\mathcal{A}$ can keep asking other queries as it wishes. Eventually, $\mathcal{A}$ may terminate and output a bit b' as its guess for b in the Test query.

The difference between static and adaptive historical tag leakage is whether HTLeak query is allowed in the second phase of the above security experiment. We give a formulation of *full corruption* (of a party) as follows, so that the partial corruption is its complement.

Full Corruption. We define the full corruption of a party $\mathcal{ID} \in \{\mathsf{id_C}, \mathsf{id_S}\}$ via a function FullC which takes as input two identities $(\mathsf{id_C}, \mathsf{id_S})$ and the number q_l of HTLeak query that is allowed, and outputs 1 to denote full corruption of $\mathcal{ID}$ and 0 otherwise. $\mathsf{FullC}(\mathcal{ID}, \mathsf{id_C}, \mathsf{id_S}, q_l) = 1$ if one of the following conditions holds:

1. $\mathsf{id_C}$ was taken as input to any $\mathsf{RegClient}$ query;
2. $cst^1_{\mathsf{id_C},\mathsf{id_S}} = cst^2_{\mathsf{id_C},\mathsf{id_S}} = \mathsf{exposed}$;
3. $cst^1_{\mathsf{id_S},\mathsf{id_C}} = cst^2_{\mathsf{id_S},\mathsf{id_C}} = cst^3_{\mathsf{id_S},\mathsf{id_C}} = \mathsf{exposed}$;
4. $cst^1_{\mathsf{id_S},\mathsf{id_C}} = cst^2_{\mathsf{id_S},\mathsf{id_C}} = \mathsf{exposed}$ and $\mathcal{A}$ queried more than q_l HTLeak queries;
5. $\mathcal{ID} = \mathsf{id_C}$ and $cst^1_{\mathsf{id_S},\mathsf{id_C}} = cst^3_{\mathsf{id_S},\mathsf{id_C}} = \mathsf{exposed}$;
6. $\mathcal{ID} = \mathsf{id_C}$, $cst^1_{\mathsf{id_C},\mathsf{id_S}} = \mathsf{exposed}$ and $\mathcal{A}$ queried more than q_l HTLeak queries.

The last two conditions are added because $\mathsf{id_C}$ has one less authentication factors than $\mathsf{id_S}$. Basically, we intend to model the authentication for a specific party $\mathcal{ID} \in \{\mathsf{id_C}, \mathsf{id_S}\}$ when $\mathsf{FullC}(\mathcal{ID}, \mathsf{id_C}, \mathsf{id_S}, q_l) = 0$.

In the following security definition, we let $\mathcal{ID}^*$ denote the party that is submitted to the Test query, and let $\widetilde{\mathcal{ID}^*}$ denote the identity that is required to provide explicit authentication. That is, $\widetilde{\mathcal{ID}^*}$ denotes $\mathsf{id_S}^*$ when $\mathsf{MN} = \mathsf{woFS}$, and $\widetilde{\mathcal{ID}^*}$ denotes either $\mathsf{id_S}^*$ or $\mathsf{id_C}^*$ when $\mathsf{MN} = \mathsf{FS}$.

Definition 2 (HMAKE Security)**.** *We say a PPT adversary* $\mathcal{A}$ $(t, \epsilon, q_l, \mathsf{MN})$*-breaks an HMAKE protocol* Π *in the security experiment with* MN*, if* $\mathcal{A}$ *runs in time* t*, and one of the following conditions is satisfied:*

- ***Authentication****: When* $\mathcal{A}$ *terminates, then with probability* ϵ *there exists an oracle* $\pi^u_{\widetilde{\mathcal{ID}^*}}$ *such that*
  - $\mathsf{FullC}(\widetilde{\mathcal{ID}^*}, \widetilde{\mathcal{ID}^*}, \mathsf{pid}^u_{\widetilde{\mathcal{ID}^*}}, q_l) = 0$ *when* $\pi^u_{\widetilde{\mathcal{ID}^*}}$ *accepts, and*
  - $\pi^u_{\widetilde{\mathcal{ID}^*}}$ *has no unique partner oracle at the party* $\mathsf{pid}^u_{\widetilde{\mathcal{ID}^*}}$*.*

  We say that $\pi^u_{\widetilde{\mathcal{ID}^*}}$ *accepts* maliciously *if it accepts satisfying the above conditions.*

- ***Key Exchange****: When* $\mathcal{A}$ *terminates and outputs a bit* b'*, and*
  - $\mathcal{A}$ *asked a* $\mathsf{Test}(\mathcal{ID}^*, u)$ *query without failure, and*
  - *if* $\mathsf{MN} = \mathsf{woFS}$ *then* $\mathsf{FullC}(\mathcal{ID}^*, \mathcal{ID}^*, \mathsf{pid}^u_{\mathcal{ID}^*}, q_l) = 0$ *and* $\mathsf{FullC}(\mathsf{pid}^u_{\mathcal{ID}^*}, \mathcal{ID}^*, \mathsf{pid}^u_{\mathcal{ID}^*}, q_l) = 0$*, and*
  - *if* $\mathsf{MN} = \mathsf{FS}$ *then* $\mathsf{FullC}(\mathcal{ID}^*, \mathcal{ID}^*, \mathsf{pid}^u_{\mathcal{ID}^*}, q_l) = 0$ *and* $\mathsf{FullC}(\mathsf{pid}^u_{\mathcal{ID}^*}, \mathcal{ID}^*, \mathsf{pid}^u_{\mathcal{ID}^*}, q_l) = 0$ *when* $\pi^u_{\mathcal{ID}^*}$ *accepts, and*
  - $\mathcal{A}$ *neither asked* $\mathsf{RevealKey}(\mathcal{ID}^*, u)$ *nor* $\mathsf{RevealR}(\mathcal{ID}^*, u)$*, and*
  - *if* $\pi^v_{\mathsf{pid}^u_{\mathcal{ID}^*}}$ *is a partner oracle of the test oracle* $\pi^u_{\mathcal{ID}^*}$*,* $\mathcal{A}$ *queried neither* $\mathsf{RevealKey}(\mathsf{pid}^u_{\mathcal{ID}^*}, v)$ *nor* $\mathsf{RevealR}(\mathsf{pid}^u_{\mathcal{ID}^*}, v)$*,*

  and then the probability b' *equals to the bit* b *sampled in the* Test *query satisfies* $|\Pr[b' = b] - 1/2| \geq \epsilon$*. We say that* $\mathcal{A}$ answers the session-key-challenge correctly *if* $b' = b$ *and all the above conditions are met.*

We say that an HMAKE protocol is $(t, \epsilon, q_l, \mathsf{MN})$*-secure, if there exists no PPT adversary that* $(t, \epsilon, q_l, \mathsf{MN})$*-breaks it.*

4.6 An Efficient HMAKE Protocol

In this section, we develop an efficient HMAKE Protocol in the random oracle model denoted by Π_{woFS}. The main construction idea of Π_{woFS} is to directly use authentication factors to derive a session key.

Protocol Description. Let $\mathbb{Z}_p$ be a cyclic group with a prime order p that has a bit-length ℓ_p, and $\mathbb{Z}_p^* = \mathbb{Z}_p/\{0\}$. In our protocol, we need a cryptographic hash function $h : \{0,1\}^* \rightarrow \mathbb{Z}_p$. We assume that the server chooses a uniform salt $\chi_{\mathsf{id_S}}$ for each client to randomize the hash function, which is implicitly used as input of h. Let ℓ_r be a bit-length defining a randomness space. In our protocols, the historical data is considered as one of the authentication factors, so we assume it to be unpredictable and have some min-entropy[3]. As stated in [28], any unpredictable string (regardless of its min-entropy) with bit-length that is larger than ℓ_p, in the random oracle model, can be used to extract an unpredictable ℓ_p-bit uniform random string in $\mathbb{Z}_p$. To avoid the leakage of historical data and tags being over the security threshold, we adopt a sliding window alike approach. We let SI be an set with size L, which stores the indices of historical data and tags that will be used for authentication and key exchange. We assume that the indices in SI can be used at most ϕ times, so once they have been use ϕ times, we will refresh SI with the next L unused historical data and tags from $(\mathcal{D}_1, \mathcal{D}_2)$.

The protocol Π_{woFS} running between a client $\mathsf{id_C}$ and a server $\mathsf{id_S}$ is shown in Fig. 4.3, which consists of three phases described below.

[3]As a validation of this assumption, we evaluated the min-entropy of sensor measurements in real industrial control systems based on one dataset of the operations of a real-world water treatment system [71]. The min-entropy of the sensor data in each stage is in the range between 4.52 and 7.80, when the system is running.

$$
\begin{array}{rcl}
\text{client } \mathsf{id_C} & & \text{server } \mathsf{id_S} \\
\hline
\multicolumn{3}{c}{\text{Initialization}} \\
sk^1_{\mathsf{id_C},\mathsf{id_S}} = mk \xleftarrow{\$} \{0,1\}^\kappa & \xrightarrow{sk^1_{\mathsf{id_C}}} & sk^1_{\mathsf{id_S},\mathsf{id_C}} := sk^1_{\mathsf{id_C},\mathsf{id_S}} \\
sk^2_{\mathsf{id_C},\mathsf{id_S}} = K \xleftarrow{\$} \mathbb{Z}_p^* & \text{secure channel} & sk^2_{\mathsf{id_S},\mathsf{id_C}} = \mathcal{D}_1 = \emptyset \\
cnt := 0 & & sk^3_{\mathsf{id_S},\mathsf{id_C}} = \mathcal{D}_2 = \emptyset \\
\hline
\multicolumn{3}{c}{\text{Tag Generation: for the } i\text{-th data}} \\
k_i := h(K \Vert i) & \xrightarrow{i, d_i, t_i} & \\
t_i := K \cdot h(d_i \Vert i) + k_i & \text{secure channel} & \text{store } (d_i) \to \mathcal{D}_1 \\
cnt := cnt + 1 & & \text{store } (i, t_i) \to \mathcal{D}_2 \\
\hline
\multicolumn{3}{c}{\text{Online Authentication and Key Exchange}} \\
\text{Sample } z \text{ distinct random indices:} & & \text{Sample } z \text{ distinct random indices:} \\
\mathsf{I}_C = (i_1, i_2, \ldots, i_z) \xleftarrow{\$} \mathsf{SI} & & \mathsf{I}_S \xleftarrow{\$} \mathsf{SI} \backslash \mathsf{I}_C \\
r_1 \xleftarrow{\$} \{0,1\}^{\ell_r} & \xrightarrow{\mathsf{I}_C, r_1} & r_2 \xleftarrow{\$} \{0,1\}^{\ell_r} \\
\mathsf{I} = \mathsf{I}_C \cup \mathsf{I}_S & \xleftarrow{\mathsf{I}_S, r_2, X, M} & \mathsf{I} = \mathsf{I}_C \cup \mathsf{I}_S \\
\mathsf{sid} := \mathsf{id_C} \Vert r_1 \Vert \mathsf{id_S} \Vert X \Vert r_2 \Vert \mathsf{I} & & \text{for } i \in \mathsf{I}\text{: } (h(d_i \Vert i), t_i) \leftarrow \mathcal{D}_1 \& \mathcal{D}_2 \\
K_{\mathsf{I}} := \sum_{i \in \mathsf{I}} h(K \Vert i) & & X := \sum_{i \in \mathsf{I}} h(d_i \Vert i) \\
Y' := K_{\mathsf{I}} + K \cdot X & & Y := \sum_{i \in \mathsf{I}} t_i \\
M' := h(mk \Vert Y' \Vert \mathsf{sid} \Vert \text{`Auth'}) & & \mathsf{sid} := \mathsf{id_C} \Vert r_1 \Vert \mathsf{id_S} \Vert X \Vert r_2 \Vert \mathsf{I} \\
\text{reject if } M \neq M' & & M := h(mk \Vert Y \Vert \mathsf{sid} \Vert \text{`Auth'}) \\
\text{accept } K_s := h(mk \Vert Y' \Vert \mathsf{sid} \Vert \text{`SeK'}) & & \text{accept } K_s := h(mk \Vert Y \Vert \mathsf{sid} \Vert \text{`SeK'})
\end{array}
$$

FIGURE 4.3: An Efficient HMAKE Protocol Π_{woFS}.

- **Initialization**. In this phase, the client $\mathsf{id_C}$ and the server $\mathsf{id_S}$ first randomly generate a symmetric authentication key $sk^1_{\mathsf{id_C},\mathsf{id_S}} = sk^1_{\mathsf{id_S},\mathsf{id_C}} := mk \xleftarrow{\$} \{0,1\}^\kappa$ which is used as the first authentication factor. The second authentication factor of $\mathsf{id_C}$ is randomly chosen as $sk^2_{\mathsf{id_C},\mathsf{id_S}} = K \xleftarrow{\$} \mathbb{Z}_p^*$, whereas the second and third authentication factors of $\mathsf{id_S}$ are initialized (temporarily) with empty sets $(sk^2_{\mathsf{id_S},\mathsf{id_C}}, sk^3_{\mathsf{id_S},\mathsf{id_C}}) = (\mathcal{D}_1, \mathcal{D}_2) = (\emptyset, \emptyset)$. However, we assume that before the protocol is running in practice, the client should generate enough authentication tokens for the server with random data via the following tag generation procedure.

- **Tag Generation**. When the client $\mathsf{id_C}$ sends a data d_i to the server $\mathsf{id_S}$, $\mathsf{id_C}$

would compute an authentication tag t_i based on $sk^2_{\mathsf{id_C},\mathsf{id_S}} = K$. Each tag is generated as $t_i := K \cdot h(d_i||i) + k_i \pmod p$, where $k_i := h(K||i)$. After the tag is generated, $\mathsf{id_C}$ would locally increase the tag counter cnt by one, and the tuple (i, d_i, t_i) is sent to $\mathsf{id_S}$ over a secure channel. Then $\mathsf{id_S}$ privately stores the tuple $(i, d_i, h(d_i||i)) \to \mathcal{D}_1$ and $(i, t_i) \to \mathcal{D}_2$, i.e., $sk^2_{\mathsf{id_S},\mathsf{id_C}}(i) = (i, d_i, h(d_i||i))$ and $sk^3_{\mathsf{id_S},\mathsf{id_C}}(i) = (i, t_i)$. Meanwhile, the secure channel might be established by out-of-band mechanism (at the initialization phase) or the session key established during the following online authentication and key exchange phase.

- **Authentication and Key Exchange Phase**. The client $\mathsf{id_C}$ and the server $\mathsf{id_S}$ would interactively run the authenticated key exchange protocol online to generate a session key K_s as shown in Fig. 4.3. The established session key will be used to protect the underlying data and tag transmission. During this phase, both parties would first respectively exchange two random nonces $r_1, r_2 \overset{\$}{\leftarrow} \{0,1\}^{\ell_r}$, and two random index selection sets $(\mathsf{I}_C, \mathsf{I}_S)$ with z distinct random indices in each set, where $\mathsf{I}_C \overset{\$}{\leftarrow} \mathsf{SI}$ and $\mathsf{I}_S \overset{\$}{\leftarrow} \mathsf{SI} \backslash \mathsf{I}_C$. Let $\mathsf{I} = \mathsf{I}_C \cup \mathsf{I}_S$. Next, $\mathsf{id_S}$ makes use of its historical data (indexed by I) to compute a message $X := \sum_{i \in \mathsf{I}} h(d_i||i) \pmod p$, and an intermediate secret $Y := \sum_{i \in \mathsf{I}} t_i \pmod p$. In our scheme, the hash values of data are not secrets. Next, Y is used as a secret seed to generate the authentication message $M := h(mk||Y||\mathsf{sid}||\text{`Auth'})$ and the final session key $K_s := h(mk||Y'||\mathsf{sid}||\text{`SeK'})$, where sid is the session identifier concatenating the protocol messages and identities of participants. The messages (X, M) are sent to $\mathsf{id_C}$ for authentication. To verify M, $\mathsf{id_C}$ computes $K_\mathsf{I} := \sum_{i \in \mathsf{I}} h(K||i) \pmod p$, $Y' := K_\mathsf{I} + K \cdot X \pmod p$, and $M' := h(mk||Y'||\mathsf{sid}||\text{`Auth'})$. If $M' \neq M$ then $\mathsf{id_C}$ rejects the session. Otherwise, it generates the session key as id_S. We assume that two parties synchronize a variable ξ which stores the times of the selection set SI that has been used. If $\xi = \phi$ then all indices in SI plus L.

Construction Discussions. To improve upon the CWZT protocol, we modify and add several critical steps to fix the vulnerabilities of the CWZT protocol and achieve the HMAKE functionality. We highlight our main differences with the state-of-the-art CWZT protocol [39] below.

- *Security improvement for authentication.* In Section 4.4, we have shown an attack to subvert the leakage resilient security property of the CWZT scheme, that an attacker who corrupts the first authentication factor and one piece of data and its tag can then steal all other secret tags. To circumvent this attack, the server in Π_{woFS} contributes a random set I_S, such that the subset of selected historical data is determined by both parties (see Fig. 4.3), instead of only relying on the client.

- *New session key exchange feature.* Unlike the CWZT protocol, our protocol realizes the full-fledged authenticated key exchange (achieving both authentication and session key security goals). Our protocol enables both parties to establish a session key for securely transmitting the new authentication factors (i.e. data and its tags), so that the historical data based authentication and key exchange make sense.

- *Other security considerations.* We consider data and its tag as distinct authentication factors, because they are stored separately. The adversary who then only corrupts either the tags or the data cannot actively impersonate as the server to the client. For instance, if the adversary does not know the data then it is unable to generate a valid X to make the client accept M. Moreover, unlike the CWZT protocol, each party should contribute a nonce r_i (for $i \in \{1, 2\}$) so that the session identifier is unique in each session to resist *replay attacks*.

- *Performance improvement.* Unlike the CWZT protocol, our protocol does not derive many session specific ephemeral keys from the first authentication factor to protect Y. Since Y is protected by a hash function in our scheme, we could simplify its computation to achieve better performance. As a result, we roughly save $3\times$ hash operations comparing to the CWZT protocol, although we provide an additional key exchange functionality.

Limitations. Nevertheless, one of the limitations of Π_{woFS} is that it cannot provide forward secrecy, when all secrets used to compute a session key K_s are compromised from either player. If the client is not fully corrupted, then along with the growth of the second authentication factor, the newly generated session key depending on the selection set (which is chosen from an increasingly larger range) can still be secure. As we will show in the security proof that the probability regarding the event: all indices of a selection set chosen in a session are compromised by the adversary before, is negligible with a proper choice of z (e.g., $z = 161$ for 128 bits security). Thus, the attacker needs to either keep stealing the second and third authentication factors or try to compromise the client's device which might be located in a more physically secure place in CPSs.

Another limitation of Π_{woFS} is that it can only satisfy static historical tag leakage. When the HTLeak query can be asked adaptively, the adversary will be able to ask HTLeak queries with indices appeared in the Test query to break the session key security. In addition, if the adversary obtained more than q_l tags, then the key exchange security is jeopardized since the session key is derived from those tags. This limitation of Π_{woFS} is caused by the side-effect of using the secret tags for both authentication and key exchange features.

Theorem 3. *Suppose that the hash function h is indistinguishable from a random oracle in time t_h and with at most q_h queries, and each data piece is unpredictable. Then Π_{woFS} is $(t', \epsilon_{\Pi_{\mathsf{woFS}}}, q_l, \mathsf{woFS})$-secure with $t' = t_h \approx t$, $\phi \le q_l$, and $\epsilon_{\Pi_{\mathsf{woFS}}} \le \frac{\rho^2}{2^{\ell_r - 1}} + 14\rho \cdot (\frac{q_l}{L-z})^z + \frac{(14\rho+22+6L)\cdot q_h}{2^{\ell_p}}$.*

Security Analysis. We divide adversaries into two categories to analyze the authentication and key exchange respectively: (i) *Authentication-adversary* can succeed in making an oracle accept maliciously; (ii) *Session-Key-adversary* is able to answer the session-key-challenge correctly.

To prove Theorem 3, we present two lemmas. Each analyzes one of the security properties of the proposed protocol. Specifically, Lemma 1 bounds the success probability ϵ_{auth} of authentication-adversaries, and Lemma 2 bounds the success probability ϵ_{skey} of session-key-adversaries. Then we have $\epsilon_{\Pi_{\mathsf{woFS}}} \le \epsilon_{\mathsf{auth}} + \epsilon_{\mathsf{skey}}$.

The full proof of Theorem 3 is given in Appendix B.1. In the following, we just present the outline of the proof.

Lemma 1. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an oracle $\Pi^u_{\mathsf{id_C}^*}$ that accepts maliciously is at most $\epsilon_{\mathsf{auth}} \le \frac{\rho^2}{2^{\ell_r}} + 6\rho \cdot (\frac{q_l}{L})^z + \frac{(6\rho+9+3L)\cdot q_h}{2^{\ell_p}}$.*

The proof of this lemma has three main steps. First, we exclude the collision among the random nonces, which occurs with negligible probability $\frac{\rho^2}{2^{\ell_r}}$ due to the birthday paradox. Let S be the set of indices that are submitted to the HTLeak query. Then, in a second step, when the third authentication factor is not corrupted (which occurs with probability 1/3 since there are 3 authentication factors), then the probability that an oracle $\pi^{u}_{\mathsf{idC}^*}$ accepts maliciously and sends out a selection set I^*_C such that $\mathsf{I}^*_C \subseteq \mathsf{S}$ is about $1 - (\Pr[\overline{\mathsf{I}^*_C \subseteq \mathsf{S}}])^\rho = 1 - (1 - (\frac{q_l}{L})^z)^\rho < \rho \cdot (\frac{q_l}{L})^z$. This implies that at least one factor of a party, which is not fully corrupted, is not known by the adversary. Hence, the adversary is only able to break the security of the protocol by its random guesses.

Lemma 2. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an adversary $\mathcal{A}$ which answers session-key-challenge correctly is at most* $\epsilon_{\mathsf{skey}} \leq \frac{\rho^2}{2^{\ell_r}} + 8\rho \cdot (\frac{q_l}{L-z})^z + \frac{(8\rho+13+3L)\cdot q_h}{2^{\ell_p}}$.

The proof of this lemma mainly relies on the authentication security and the compromised secret tags. The key issue here is whether the adversary knows all secret tags used to compute the session key of the test oracle. Note that the adversary can only manipulate the selection set of the client I_C which is not authenticated. Hence, the probability that the selection set I^*_S used by the test oracle such that $\mathsf{I}^*_S \subseteq \mathsf{S}$ is about $\rho \cdot (\frac{q_l}{L-z})^z$ which can be negligible with proper parameters.

4.7 A HMAKE Protocol with Stronger Security

In this section, we propose an HMAKE protocol called Π_{FS}, which overcomes the limitations of Π_{woFS}. The idea of the construction of this protocol is to make use of the authentication procedure as a compiler to transform a general passively secure two-message key exchange protocol to achieve HMAKE security. To realize our idea,

client $\mathsf{id_C}$		server $\mathsf{id_S}$
The **Initialization** and **Tag Generation** phases are identical to those of Π_{woFS}		
Online Authentication and Key Exchange		
Sample z distinct random indices:		Sample z distinct random indices:
$\mathsf{I}_C = (i_1, i_2, \ldots, i_z) \xleftarrow{\$} \mathsf{SI}$		$\mathsf{I}_S \xleftarrow{\$} \mathsf{SI} \backslash \mathsf{I}_C$
$\tilde{r_1} \xleftarrow{\$} \{0,1\}^{\ell_r}$		$\tilde{r_2} \xleftarrow{\$} \{0,1\}^{\ell_r}$
$m_1 \leftarrow \mathsf{TKE.MSG}(\mathsf{id_C}, \tilde{r}_1, \emptyset)$	$\xrightarrow{\mathsf{I}_C, m_1}$	$m_2 \leftarrow \mathsf{TKE.MSG}(\mathsf{id_S}, \tilde{r}_2, m_1)$
$\mathsf{I} = \mathsf{I}_C \cup \mathsf{I}_S$	$\xleftarrow{\mathsf{I}_S, X, m_2, M_2}$	$\mathsf{I} = \mathsf{I}_C \cup \mathsf{I}_S$
$\mathsf{sid} := \mathsf{id_C} \Vert \mathsf{id_S} \Vert X \Vert m_1 \Vert m_2 \Vert \mathsf{I}$		for $i \in \mathsf{I}$: $(h(d_i \Vert i), t_i) \leftarrow \mathcal{D}_1 \& \mathcal{D}_2$
$K_{\mathsf{I}} := \sum_{i \in \mathsf{I}} h(K \Vert i)$		$X := \sum_{i \in \mathsf{I}} h(d_i \Vert i)$
$Y' := K_{\mathsf{I}} + K \cdot X$		$Y := \sum_{i \in \mathsf{I}} t_i$
$M_2' := h(mk \Vert Y' \Vert \mathsf{sid} \Vert \text{'Auth'} \Vert 2)$		$\mathsf{sid} := \mathsf{id_C} \Vert \mathsf{id_S} \Vert X \Vert m_1 \Vert m_2 \Vert \mathsf{I}$
reject if $M_2 \neq M_2'$		$M_2 := h(mk \Vert Y \Vert \mathsf{sid} \Vert \text{'Auth'} \Vert 2)$
$K^{ke} \leftarrow \mathsf{TKE.SKG}(\mathsf{id_C}, \tilde{r}_1, \mathsf{id_S}, m_2)$		$M_1' := h(mk \Vert Y \Vert \mathsf{sid} \Vert \text{'Auth'} \Vert 1)$
$M_1 := h(mk \Vert Y \Vert \mathsf{sid} \Vert \text{'Auth'} \Vert 1)$	$\xrightarrow{M_1}$	reject if $M_1 \neq M_1'$
		$K^{ke} \leftarrow \mathsf{TKE.SKG}(\mathsf{id_S}, \tilde{r}_2, \mathsf{id_C}, m_1)$
accept $K_s := h(K^{ke} \Vert \mathsf{sid})$		accept $K_s := h(K^{ke} \Vert \mathsf{sid})$

FIGURE 4.4: An HMAKE Protocol Π_{FS} with Perfect Forward Secrecy.

we need to add one more authentication message to achieve mutual explicit authentication for both parties. Comparing with Π_{woFS}, the protocol Π_{FS} can achieve not only PFS but also the resilience of adaptive historical tag leakage. Also, Π_{FS} can still guarantee authentication and key exchange security when *all* tags are corrupted but the historical data is not corrupted. It is because that the session key in Π_{FS} does not depend on the tags anymore.

Protocol Description. In this protocol, one more primitive is needed, i.e. a TKE protocol with parameters $pms \leftarrow \mathsf{TKE.Setup}(1^{\kappa})$. We assume that the randomness space of TKE is $\mathcal{R}_{\mathsf{TKE}} = \{0,1\}^{\ell_r}$. We depict the protocol Π_{FS} in Fig. 4.4.

Theorem 4. *Suppose that the hash function h is indistinguishable from a random oracle in time t_h and with at most q_h queries, and each data piece is unpredictable, and the two-message key exchange protocol* TKE *is* $(t_{\mathsf{TKE}}, \epsilon_{\mathsf{TKE}})$*-passively-secure. Then*

Π_{FS} *is* $(t', \epsilon_{\Pi_{\mathsf{FS}}}, q_l, \mathsf{FS})$*-secure with* $t' = (t_{\mathsf{TKE}} + t_h) \approx t$*,* $\phi \leq q_l$*, and* $\epsilon_{\Pi_{\mathsf{FS}}} \leq \frac{\rho^2}{2^{\ell_r - 1}} + 18\rho \cdot (\frac{q_l}{L-z})^z + \frac{(18+6L+18\rho)\cdot q_h}{2^{\ell_p}} + 2\rho \cdot (2\rho + 2) \cdot \epsilon_{\mathsf{TKE}}$.

Similarly, we prove Theorem 4 via the following two lemmas.

To prove Theorem 4, we present two lemmas. Lemma 3 bounds the success probability ϵ_{auth} of authentication-adversaries, and Lemma 4 bounds the success probability ϵ_{skey} of session-key-adversaries. Then we have $\epsilon_{\Pi_{\mathsf{FS}}} \leq \epsilon_{\mathsf{auth}} + \epsilon_{\mathsf{skey}}$.

In the following, we just present the outline of the proof.

Lemma 3. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an oracle $\Pi^u_{\mathcal{ID}^*}$ that accepts maliciously is at most* $\frac{\rho^2}{2^{\ell_r}} + 2\rho \cdot \epsilon_{\mathsf{TKE}} + 9\rho \cdot (\frac{q_l}{L-z})^z + \frac{(9+3L+9\rho)\cdot q_h}{2^{\ell_p}}$.

Lemma 4. *For any adversary $\mathcal{A}$ running in time $t' \approx t$, the probability that there exists an adversary $\mathcal{A}$ which answers session-key-challenge correctly is at most* $\frac{\rho^2}{2^{\ell_r}} + 9\rho \cdot (\frac{q_l}{L-z})^z + \frac{(9+3L+9\rho)\cdot q_h}{2^{\ell_p}} + 2\rho \cdot (2\rho + 1) \cdot \epsilon_{\mathsf{TKE}}$.

Basically, the proof of this theorem can be extended from the proof of Theorem 3. We outline our proof idea as follows. In contrast to Π_{woFS}, Π_{FS} can provide mutual explicit authentication. The authentication message M_1 sent from the client is computed in a similar way as M in Π_{woFS} and M_2 in Π_{woFS}, therefore we can reduce the authentication security regarding M_1 in a similar way as the proof of Theorem 3 when the tags leakage is below a threshold. The advantage of an adversary $\mathcal{A}$ breaking the authentication of Π_{FS} is twice of breaking the authentication of Π_{woFS}. Also, the random values r_1 and r_2 in Π_{woFS} are replaced with m_1 and m_2 in Π_{FS}, because of the security of the TKE protocol [130, Lemma1].

Moreover, if there is no adversary that can break the authentication property of Π_{FS}, then there would be only passive adversary between the test oracle and its partner oracle (which must exist due to the explicit authentication messages M_1 and M_2). This fact enables us to reduce the key exchange security of Π_{FS} to the security of TKE. We present the specific security reduction in Appendix B.2.

4.8 Security Enhancement for Legacy Devices

In this section, we show an important practical aspect of our HMAKE protocols, i.e. they are able to strengthen the security of existing legacy devices without modifying them.

Here we consider a legacy device that has a symmetric key mk shared with the server (i.e., the first authentication factor in our scheme)[4]. A common (toy) AKE solution deployed on a legacy device might be like that two parties generate the session key (or the authentication message) in a form $K_s := h(mk, r_C||r_S||aux)$, where r_C and r_S are nonces selected by the client and the server respectively (in the toy AKE scheme), and aux may contain other protocol messages if any (e.g., Diffie-Hellman public keys). Our HMAKE protocols can be simply adapted to enhance the security of such a legacy device with the above toy AKE without modifying its original operations. To deploy our protocol, a separate secure device, storing the tag key K of the client, is directly and securely connected to the legacy device (e.g., via local LAN cables). After the new device executes our HMAKE protocol steps except the session key generation, it only needs to send the secret hash value $H(Y||sid||\text{`SeK'})$ to the legacy device as the r_S in the toy AKE scheme. The server can compute the

[4]In case the legacy devices do not have an AKE built in, it becomes trivial for us to enhance their security. We can simply add a new device like what the authors did in [35] to intercept the traffic of legacy devices and run the complete HMAKE protocols with the server. This is still legacy-compliant. However, the practical difficulty is how to be compatible with legacy devices which run common AKE protocols.

same session key in the exactly same way. Meanwhile, we can choose to drop the explicit authentication message M in our protocol depending on whether the legacy protocol has explicit message authentication steps[5].

To apply the above security enhancement in practice, we only need to check whether the legacy device runs an AKE protocol (or its variant – Authenticated Confidential Channel Establishment [131]) in the above form of toy example. One famous protocol instance meeting our requirement is the Transport Layer Security (TLS) Protocol with pre-shared key cipher-suits [50, 163, 131, 53] which are proposed for power-constrained devices (such as EMV card [158]). For example, our first protocol Π_{woFS} can be used to enhance the security of TLS_PSK, and the second protocol Π_{FS} is suitable for TLS_DHE_PSK, where TLS_PSK uses only symmetric key (PSK) for authentication, and TLS_DHE_PSK uses a Diffie-Hellman exchange authenticated with a pre-shared key. Besides, the TLS protocols have explicit authentication steps.

4.9 Comparison

In this section, we briefly compare our proposed schemes with recent typical lightweight authenticated key exchange protocols, i.e., Das et al., [49], He et al. [82] and Challa et al. [37], just for reference. Although these protocols are designed for the three-party case, two-party AKE procedure is also involved. We compare these four protocols from the following perspectives: (i) authentication factors, (ii) main security properties, (iii) number of communication passes, and (iv) computation cost. To compare

[5]The CWZT scheme is not legacy-compliant since the computation of Y needs two authentication factors, so it should be deployed in one device where both authentication factors are stored together.

Table 4.1: Comparison

Protocol	Auth Factors	Security Properties				Pass	Computation
		S-Auth	M-Auth	B-Leak	PFS		
Das et al.[49]	Bio+PW +LSK	$\surd$	$\times$	$\times$	$\surd$	2	31H+1FE+4MUL
He et al.[82]	PW+LSK	$\surd$	$\times$	$\times$	$\surd$	2	21H+4MUL
Challa et al. [37]	Bio+PW+LPK	$\surd$	$\times$	$\times$	$\times$	2	1Fe+14Mul+12h
Π_{woFS}^{RO}	LSK+HD	$\surd$	$\times$	$\surd$ (static)	$\times$	2	326 H
Π_{FS}^{RO}	LSK+HT+HD	$\surd$	$\surd$	$\surd$ (adaptive)	$\surd$	3	328H+4MUL

the computational cost, we instantiate our protocol Π_{FS}^{RO} with the elliptic curve cryptography (ECC) based Diffie-Hellman key exchange protocol (as the other compared protocols). Furthermore, we let 'FE' denote a fuzzy extractor operation to obtain a secret from biometrics. We let 'S-Auth' denote single-side explicit authentication, 'M-Auth' denote mutual authentication, 'B-Leak' denote bounded leakage. To compare the computation cost, let 'H' denote hash function operation and 'MUL' denote an ECC multiplication. Let 'Bio' denote the biometric authentication factor, 'PW' denote password, 'HD' denote historic data, 'HT' denote historic tags, 'LSK' denote long-term symmetric key, and 'LPK' denote the long-term public key.

We summarize the comparison in Table 4.1.

Though our protocols are less efficient than Das et al. and He et al. protocols, we provide one more security property, i.e., bounded-leakage resilience. Since a hash operation is not expensive, the overall performance of Π_{woFS} is still practical (as shown in Table 4.2) for constrained devices.

4.10 Implementation and Experimental Results

Implementation Parameters. We consider the upper-bound of the sessions of each party to be $\rho = O(2^{30})$ in practice, $\frac{q_l}{L-z} \approx 1/2$, $q_h = 2^{30}$ and $L = 2^{15}$. In the following, we list the parameters used in our implementation of Π_{woFS} and Π_{FS} based on the corresponding security levels: (i) for the security level $\kappa = 80$, we use $\ell_r = 141$, $z = 113$, $\ell_p = 145$ for Π_{woFS}, and $\ell_p = 224$ for Π_{FS}; (ii) for the security level $\kappa = 128$, we use $\ell_r = 189$, $z = 161$, $\ell_p = 193$ for Π_{woFS}, and $\ell_p = 320$ for Π_{FS}.

Experiments Setup. We used one PC (with Intel Core i7-8750H processor) as a server, and a Raspberry Pi 3 (with Quad Core 1.2GHz Broadcom BCM2837 CPU and 1GB RAM) is taken as a client. Our implementation is based on MIRACL cryptographic library [152], where the hash function used is SHA256 in Π_{woFS} and SHA384 in Π_{FS} , and the TKE protocol used in our second protocol is the Diffie-Hellman key exchange protocol based on the standard elliptic curve (over $GF(p)$) provided by MIRACL.

Performance Evaluation. We first measured the tag generation time on the client. It takes 0.55 ms per tag, assuming data size is 1KB. Also, we measured the time consumed by the authentication protocol and the key generation procedure separately on both the server and the client. The performance is reported in milliseconds in Table 4.2; 'KE' denotes the time for ephemeral key and the session key generations, and 'Auth' denotes the performance of all other steps in authentication. The performance bottleneck is clearly on the client side, because it is a resource-constrained embedded system device, and it needs $2z$ hash operations for one authentication. However, even for 128 bits security, the performance of the client in Π_{woFS} is only 24.695 ms, which is efficient enough to be deployed in real-world applications.

	Π_{woFS}		Π_{FS}	
	Server	Client	Server	Client
Auth	0.137/0.213	17.184/24.336	1.986/4.056	65.561/82.795
KE	0.030/0.045	0.299/0.359	1.827/3.879	54.530/69.842

Table 4.2: The performance of the proposed HMAKE protocols for (80-bit security/128-bit security), measured in ms.

4.11 Conclusions and Open Problems

In this paper, we have shown two ways to build multi-factor AKE protocols based on historical data in the random oracle model. The proposed protocols are efficient enough for resource-constrained devices in CPS or IoT. In particular, the first protocol only requires a few hash operations on the client. One open problem worth solving in the future is how to construct a HMAKE protocol in the standard model. Its challenge is to generate a pseudo-random seed from the authentication tags.

Chapter 5

Summary

In this dissertation, we first defined a formal framework for characterizing different types of adversaries in cyber-physical systems. Using the framework, we can reason about the security of a CPS and identify possible solutions. As concrete examples, one privacy-preserving fault-tolerant sensor fusion system is presented to prevent pollution attacks while persevering the privacy of individual users. In addition, a secure logging and intrusion detection system for industrial control systems is presented. Since it is implemented on the firmware or hardware layer, it can be secure against an attack like Stuxnet malware originated from the SCADA system. Lastly, we introduced two novel multi-factor authenticated key exchange protocols, which use the historical data stored in the SCADA server as an additional authentication factor.

Appendix A

Alternative Approaches, Correctness Proof and Additional Examples for PwoP

A.1 Alternative Approaches to Intrusion-Tolerant and Privacy-Preserving Sensor Fusion

We describe two alternative approaches to intrusion-tolerant and privacy-preserving sensor fusion, and compare them with PwoP.

A.1.1 Order-Preserving Encryption Based Approach

Order-preserving encryption (OPE) [25, 26, 159, 108] is an encryption scheme where the order of ciphertexts matches that of the corresponding plaintexts. OPE is a powerful primitive most known to enable performing a large class of queries on encrypted databases. However, OPE is also suggested for use in encrypted data aggregation in sensor networks [189] and multimedia content protection [60].

We observe that non-interactive, deterministic OPE [25, 26] may be used to achieve the goal of fault-tolerant and privacy-preserving sensor fusion, yet for a limited class of problems, and with a much weak security guarantee. For instance, we consider how to do this for M-g-U: Initially, assume all sensors and the client share a group OPE key. Using this group key, each user encrypts only two values, i.e., the leftmost and rightmost endpoint of its input interval. Since all the encrypted values reveal their order information, the server is able to run the fault-tolerant sensor averaging algorithm in "plaintexts," and returns the leftmost and rightmost endpoint of the resulting interval to the client. Then the client can use the group key to decrypt the two ciphertexts.

The above construction is simple, but suffers from several problems. First, the construction leaks all the order information. For many applications, the order information is exactly what one strives to protect. Even worse, all existing non-interactive OPEs leak more than just the order of the values [159]. Correspondingly, even with the non-collusion assumption between the server and sensors, and allowing leaking all the order information, it is difficult, if not impossible, to prove the construction secure against even a semi-honest server under (any) appropriate simulation-based definition of security. Further, it is also unclear how to achieve integrity (i.e., ensure the server to faithfully return the client correct OPE values, rather than arbitrary values). Last, the construction can only apply to the sensor fusion algorithms with unbounded accuracy, but we do not know how to deal with the ones with bounded accuracy.

Interactive, ideal-secure OPEs that reveal no additional information on the plaintexts besides their order do exist [159, 108]. The interactive nature of these schemes, however, makes them ill-suited for our setting where some sensors may want to learn more information about other sensors.

A.1.2 Set Representation Based Approach

We extend the idea of multi-party set representation (SR) [101] to provide solutions for *some* (but not all) of the fault-tolerant fusion algorithms.

The idea for the set representation method is as follows: To compute the resulting interval, one may simply approximate the *real* interval with as many discrete elements as possible. Let σ denote the number of elements in the entire universe. The idea can naturally lead to an algorithm with $O(n\sigma)$ time.

Before proceeding to our findings, let's briefly describe the server-aided private set intersection protocol by Kamara, Mohassel, Raykova, and Sadeghian (KMRS) [102]. Let S denote the set of party p_i. All the parties who have private inputs should first jointly generate a secret key for a pseudorandom permutation (PRP) E. Then each party randomly permutes the set $\mathsf{E}_k(S)$, and sends the permuted set to the server, which simply computes and returns the intersection of all the encrypted sets. The protocol above is secure with a semi-honest server or any collusion of malicious parties. Further techniques were developed by KMRS to ensure the protocol to remain secure against a malicious server.

We may base the idea to build a privacy-preserving and fault-tolerant scheme. Compared to the GC based approach, set representation based one has much lower client to server communication complexity, but much larger communication complexity and time complexity between sensors and the server. The property makes SR based approach not suitable for applications where sensors has limited bandwidth and computational power. Meanwhile, SR based approach only works for semi-honest sensors. (The reason is that a malicious sensor might not provide consecutive encrypted data.) This assumption is hard to justify, as the number of the sensors may be large and the sensors are distributed in different locations. Last, SR based approach only works for a rather limited set of fault-tolerant algorithms.

A.2 Correctness Proof

Theorem 5. *In the circuit designs of M-g-U, M-g, M-g-m, ChM-dD and ChM-dD-sso, only $g+1$ positions out of $2n$ can possibly have a prefix sum equal $n-g$.*

Proof: Let $z_1, z_2, z_3, ...z_{2n}$ be an array of sign values (in the form of ± 1) of a sorted array in an ascending order generated by our modified sorting network described in Sec. 2.6. We denote the prefix sum by $A_j = \sum_{i=1}^{j} z_i$.

To prove the theorem, we just need to show the following two claims are correct:

1. If $\exists j$ such that $A_j = n-g$, then only A_{j+2k} can possibly equal $n-g$, where k is an integer and $1 \leq j+2k \leq 2n$.

2. If $A_j = n-g$, then $n-g \leq j \leq n+g$.

Indeed, given the two claims, we can find that there are in total $g+1$ positions that can possibly have a prefix sum equal $n-g$. They are $n-g+2h$, where h is an integer and $0 \leq h \leq g$. The theorem will then follow.

Proof of claim 1: Suppose $A_j = n-g$. Let us consider the set of z_i for $1 \leq i \leq j$. Since z_i can only be ± 1, we can only replace $+1$ with -1 or replace -1 with $+1$ to change A_j. Therefore A_j can only be $n-g+2k$, where k is an integer and $0 \leq n-g+2k \leq n$.

This also implies that it is impossible for A_{j-1} and A_{j+1} to equal $n-g$; they can only possibly be $n-g+1+2k$.

Proof of claim 2: Since $A_j = \sum_{i=1}^{j} z_i$, and z_i can only be ± 1, the smallest index j, such that $A_j = n-g$, is $n-g$. Similarly, the largest index j, such that $A_j = n-g$, is $2n-(n-g) = n+g$.

Theorem 6. *In the circuit designs of M-g-U, M-g, M-g-m, M-op, ChM-dD and ChM-dD-sso, modified sorting networks and index select are only needed once.*

Proof: We use the same notation as above. To compute the minimum value (left end) of the resulting interval, we need to compute the prefix sum $A_j = \sum_{i=1}^{j} z_i$, and find the leftmost position j such that $A_j = n-g$. Similarly, to compute the maximum value of the resulting interval, we need to compute the postfix sum $B_j = \sum_{i=j}^{2n} z_i$, and find the rightmost position j such that $B_j = -(n-g)$. Notice that all z_i's come in pairs of $+1$ and -1, so the sum of all z_i's must be 0. This implies that for any j, $A_j + B_{j+1} = \sum_{i=1}^{2n} z_i = 0$. Thus we can directly obtain the array of B_j from A_j without performing the addition operations again. This saves an additional copy of modified sorting network and index select, including prefix or postfix sum and equality checkers (no equality checkers in the circuit of M-op, since g is unknown), for computing the right end of the resulting interval. Since $B_j = -A_{j-1}$, when we apply the sorted array of sensor inputs to the max value max index module, we need to left shift the array of sensor inputs for one position. Note that bit shifting is completely free in circuits.

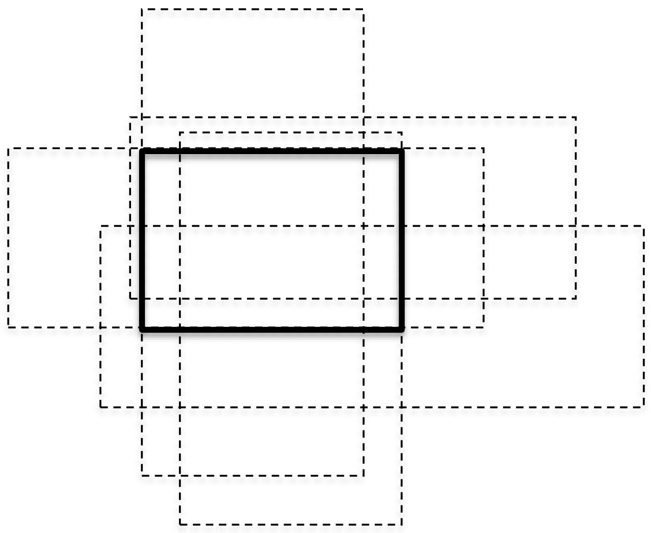


FIGURE A.1: Example of ChM-dD with $d = 2$. Five rectangles with dashed lines are the input rectangles, and the rectangle with solid lines is the aggregated rectangle.

A.3 Example for Multidimensional Algorithms

Figure. A.1 shows an example of ChM-2D algorithm for a five-sensor system. The five rectangles with dashed lines are the input rectangles from five sensors. We run a M-g algorithm for both dimensions, leading to two resulting intervals. These two intervals construct the final resulting rectangle (solid block in Figure. A.1).

Appendix B

Security Proofs for HMAKE

B.1 Proof of Theorem 3

The proof of Theorem 3 has two parts: (i) the proof of Lemma 1 for authentication security, and (ii) the proof of Lemma 2 for key exchange security.

B.1.1 Proof of Lemma 1

In the following, we show the proof of Lemma 1 in a sequence of games. Let $\mathsf{S}_i^{\mathrm{auth}}$ denote an event that there exists an authentication-adversary wins in Game i.

Game 0. This game equals the real security experiment described in Section 4.5. Meanwhile, all oracle queries are answered honestly according to our protocol specification. Thus, we have that

$$\Pr[\mathsf{S}_0^{\mathrm{auth}}] = \epsilon_{\mathsf{auth}}.$$

Game 1. In this game, the challenger $\mathcal{C}$ proceeds exactly like the previous game, but adds an abort rule to all Send queries that it aborts if: two oracles generate the same nonce (i.e., either r_1 or r_2). Note that there are ρ oracles at each honest party. By applying the birthday paradox, we have that

$$\Pr[\mathsf{S}_0^{\text{auth}}] \leq \Pr[\mathsf{S}_1^{\text{auth}}] + \frac{\rho^2}{2^{\ell_r}}.$$

Due to the modification in this game, each session identifier sid including $r_1||r_2$ is uniquely shared with its partner oracle. The unique sid ensures the uniqueness of each authentication message M generated involving sid is unique as well (even though the selection set I has collision).

Game 2. Note that the adversary can choose to corrupt either the first authentication factor or the second authentication factor, but not both of them. Hence we need to proceed with the proof following one of the following corruption cases:

- **Corruption Case 1**: $\mathcal{A}$ did not ask any $\mathsf{Corrupt1}(\cdot)$ query;
- **Corruption Case 2**: $\mathcal{A}$ did not ask any $\mathsf{Corrupt2}(\cdot)$ query;
- **Corruption Case 3**: $\mathcal{A}$ did not ask any $\mathsf{Corrupt3}(\cdot)$ query, and $\mathcal{A}$ asked no more than q_l queries to $\mathsf{HTLeak}(3, \cdot)$.

In this game, $\mathcal{C}$ guesses which the above corruption case would occur. If $\mathcal{C}$ guesses incorrectly, then it halts the game. The probability that $\mathcal{C}$ succeeds in guessing the corruption case is bounded by $1/3$. Thus, we have that

$$\Pr[\mathsf{S}_1^{\text{auth}}] = 3 \cdot \Pr[\mathsf{S}_2^{\text{auth}}].$$

Game 3. Let S be the set of indices that are submitted to the HTLeak query. In this game, when the corruption case 2 or case 3 occurs, then we add an abort rule: $\mathcal{C}$

aborts if there is an oracle $\pi^u_{\mathsf{id_C}^*}$ which accepts maliciously and sends out a selection set I^*_C such that $\mathsf{I}^*_C \subseteq \mathsf{S}$ (which means all secrets associated with indices in I^*_C are leaked). Note that the size of I^*_C is z and the size of S is $q_l >> z$. We bound the probability

$$\Pr[\mathsf{I}^*_C \subseteq \mathsf{S}] = \frac{\binom{q_l}{z}}{\binom{L}{z}} < (\frac{q_l}{L})^z.$$

Let $\mathsf{abort}^{auth}_{MS}$ denote the event that $\mathcal{C}$ aborts in this game. Since there are at most ρ oracles at $\mathsf{id_C}^*$ we have that

$$\begin{aligned}\Pr[\mathsf{abort}^{auth}_{MS}] &= 1 - (\Pr[\overline{\mathsf{I}^*_C \subseteq \mathsf{S}}])^\rho \\ &= 1 - (1 - (\frac{q_l}{L})^z)^\rho < \rho \cdot (\frac{q_l}{L})^z,\end{aligned}$$

with sufficient large z. Therefore, we have that

$$\Pr[\mathsf{S}^{\text{auth}}_2] = \Pr[\mathsf{S}^{\text{auth}}_3] + \Pr[\mathsf{abort}^{auth}_{MS}] < \Pr[\mathsf{S}^{\text{auth}}_3] + \rho \cdot (\frac{q_l}{L})^z.$$

Note that here we only consider the selection set I^*_C not I_S since I_S might be chosen by the adversary in an impersonation attack. If $\mathcal{C}$ does not abort in this game, then it implies that each session must choose a I^*_C containing at least one uncompromised index when most of the secret tags are uncompromised.

Game 4. In this game, $\mathcal{C}$ aborts if one of the uncompromised secrets (which could be any factors) is asked by the adversary $\mathcal{A}$ in a random oracle query. This implies $\mathcal{A}$ knows the uncompromised secret. Note that the input of each hash operation is unique (by assumptions) that would result in a unique random hash value. To learn an uncompromised secret, an adversary may test many random oracle queries with its own inputs. Specifically, $\mathcal{C}$ aborts if and only if one of the following condition holds:

- When the corruption case 1 occurs, $sk^1_{\mathsf{id_C}^*,\mathsf{id_S}^*}$ is asked by $\mathcal{A}$ in a random oracle query;

- When the corruption case 2 occurs, either K of $\mathsf{id_C}^*$ or one of the uncompromised data pieces is asked by $\mathcal{A}$ in a random oracle query.

- When the corruption case 3 occurs, either K of $\mathsf{id_C}^*$ or one of the uncompromised tags is asked by $\mathcal{A}$ in a random oracle query.

Since each data piece is not known by the adversary (under the corruption case 2) and unpredictable, and all the confidential secretes $\{K, d_i, sk^1_{\mathsf{id_C}^*,\mathsf{id_S}^*}\}$ are chosen uniformly at random with bit-length at least ℓ_p, and $\mathcal{A}$ can only guess them with q_h trials in conjunction with her random oracle queries. Thus we have that

$$\Pr[\mathsf{S}_3^{\text{auth}}] = \Pr[\mathsf{S}_4^{\text{auth}}] + \frac{(3+L)q_h}{2^{\ell_p}}.$$

Game 5. $\mathcal{C}$ proceeds this game exactly as before, but aborts if an oracle $\pi^u_{\mathsf{id_C}^*}$ such that $\mathsf{FullC}(\mathsf{id_C}^*, \mathsf{pid}^u_{\mathsf{id_C}^*}) = 0$ (throughout the game) received an $X^u_{\mathsf{id_C}^*}$ which is not sent from its partner oracle. As each selection set SI is only used for $\phi \leq q_l$ times, the maximum hashed data $h(d_i||i)$ leaked from X is bound to q_l. With the similar argument in the Game 3, we have that the $X^u_{\mathsf{id_C}^*}$ should be computed involving a secret value $h(d^*_i||i^*)$ that is not compromised (under the corruption case 2) with probability $\rho \cdot (\frac{q_l}{L})^z$. Since $X^u_{\mathsf{id_C}^*}$ is computed based on the distinct selection set SI and uniform random hash values (due to the random oracle queries with unique inputs), each $X^u_{\mathsf{id_C}^*}$ is unique as well. So that $X^u_{\mathsf{id_C}^*}$ cannot be forged or replayed with non-negligible probability. Namely, $\mathcal{A}$ can only randomly guess $X^u_{\mathsf{id_C}^*}$. Hence, we have that

$$\Pr[\mathsf{S}_4^{\text{auth}}] \le \Pr[\mathsf{S}_5^{\text{auth}}] + \rho \cdot (\frac{q_l}{L})^z + \frac{\rho \cdot q_h}{2^{\ell_p}}.$$

Game 6. In this game, $\mathcal{C}$ aborts if $\mathcal{A}$ asked a random oracle query with the value $Y^u_{\mathsf{id_C}^*}$ of an oracle $\pi^u_{\mathsf{id_C}^*}$ such that $\mathsf{FullC}(\mathsf{id_C}^*, \mathsf{pid}^u_{\mathsf{id_C}^*}) = 0$ throughout the game. Recall that $Y^u_{\mathsf{id_C}^*}$ should be computed involving a secret tag t_{i^*} with index i^* that has not been submitted to the HTLeak query. Furthermore, $Y^u_{\mathsf{id_C}^*}$ is hidden by the hash function. Hence, $\mathcal{A}$ who does not know t_{i^*} cannot compute $Y^u_{\mathsf{id_C}^*}$ (respectively) due to the modification in the previous game. So that $\mathcal{A}$ can only randomly guess $Y^u_{\mathsf{id_C}^*}$. Analogously, we have that

$$\Pr[\mathsf{S}_5^{\text{auth}}] = \Pr[\mathsf{S}_6^{\text{auth}}] + \frac{\rho \cdot q_h}{2^{\ell_p}}.$$

Game 7. In this game, for each oracle $\pi^u_{\mathsf{id_C}^*}$ such that $\mathsf{FullC}(\mathsf{id_C}^*, \mathsf{pid}^u_{\mathsf{id_C}^*}, q_l) = 0$, $\pi^u_{\mathsf{id_C}^*}$ rejects if it receives a message which is not sent by its partner oracle having a matching conversation to $\pi^u_{\mathsf{id_C}^*}$. Since $\mathcal{A}$ cannot compute $Y^u_{\mathsf{id_C}^*}$ used by $\pi^u_{\mathsf{id_C}^*}$ for verification, it is unable to distinguish this game from the previous game. Thus the advantage of $\mathcal{A}$ in this game is zero.

Summing up the probabilities in all the above games, we have the result of Lemma 1.

B.1.2 Proof of Lemma 2

Let S_i^{ke} denote an event that there exists a session-key-adversary answers the session-key-challenge correctly in Game i. The proof of this lemma is quite similar to the proof of Lemma 1. We may omit some similar details to avoid repetition. We show the proof of this lemma by the following games.

Game 0. This game equals the real security experiment described in Section 4.5. We have that

$$\Pr[\mathsf{S}_0^{\mathrm{ke}}] - 1/2 = \epsilon_{\mathsf{skey}}.$$

Game 1. In this game, $\mathcal{C}$ aborts if the owner of the test oracle $\pi^u_{\mathcal{ID}^*}$ such that $\mathcal{ID}^* = \mathsf{id_C}^*$, $\mathsf{FullC}(\mathcal{ID}^*, \mathsf{pid}^u_{\mathcal{ID}^*}, q_l) = 0$, and $\pi^u_{\mathcal{ID}^*}$ accepts without a partner oracle at $\mathsf{pid}^u_{\mathcal{ID}^*}$. Due to the authentication property of the protocol, we have that

$$\Pr[\mathsf{S}_0^{\mathrm{ke}}] = \Pr[\mathsf{S}_1^{\mathrm{ke}}] + \epsilon_{\mathsf{auth}}.$$

Hence, if the owner of the test oracle is the honest client then it must have a matching conversation at the server.

Game 2. In this game, $\mathcal{C}$ would guess in advance which corruption case will occur to the test oracle and its partner oracle. Note that the corruption case 3 will never occur by the security definition. $\mathcal{C}$ aborts if it guesses incorrectly. Thus we have

$$\Pr[\mathsf{S}_1^{\mathrm{ke}}] = 2 \cdot \Pr[\mathsf{S}_2^{\mathrm{ke}}].$$

Game 3. Recall that S is assumed to be the set of indices that are submitted to the HTLeak query. We add an abort rule: $\mathcal{C}$ aborts if the test oracle's owner is $\mathsf{id_S}^*$ and $\pi^u_{\mathsf{id_S}^*}$ sends out a selection set I^*_S such that $\mathsf{I}^*_S \subseteq \mathsf{S}$. Note that the I^*_S is chosen from $\mathsf{SI} \backslash \mathsf{I}_{C^*}$, where I_{C^*} is the selection set received by $\pi^u_{\mathsf{id_S}^*}$ (that may be chosen by $\mathcal{A}$). Similarly, we bound the probability

$$\Pr[\mathsf{I}^*_S \subseteq \mathsf{S}] = \frac{\binom{q_l}{z}}{\binom{L-z}{z}} < (\frac{q_l}{L-z})^z.$$

Let abort^{ke}_{MS} denote the event that $\mathcal{C}$ aborts in this game. Therefore, we have

$$\Pr[\mathsf{S}^{\mathrm{ke}}_2] = \Pr[\mathsf{S}^{\mathrm{ke}}_3] + \Pr[\mathsf{abort}^{ke}_{MS}] < \Pr[\mathsf{S}^{\mathrm{ke}}_3] + \rho \cdot (\frac{q_l}{L-z})^z.$$

Game 4. In this game, $\mathcal{C}$ aborts if one of the uncompromised secrets is asked by the adversary $\mathcal{A}$ in a random oracle query. Thus we have that

$$\Pr[\mathsf{S}^{\mathrm{ke}}_3] = \Pr[\mathsf{S}^{\mathrm{ke}}_4] + \frac{2q_h}{2^{\ell_p}}.$$

Game 5. In this game, $\mathcal{C}$ aborts if $\mathcal{A}$ asked a random oracle query with the value $Y^u_{\mathcal{ID}^*}$ of the test oracle $\pi^u_{\mathcal{ID}^*}$ such that $\mathsf{FullC}(\mathcal{ID}^*, \mathsf{pid}^u_{\mathcal{ID}^*}) = 0$ (throughout the game). Therefore, we have that

$$\Pr[\mathsf{S}^{\mathrm{ke}}_4] = \Pr[\mathsf{S}^{\mathrm{ke}}_5] + \frac{\rho \cdot q_h}{2^{\ell_p}}.$$

Game 6. We replace the session key of the test oracle and its partner oracle (if any) with a truly random key that is independent of the bit chosen by the test oracle. Thus the adversary gains no advantage in this game, i.e.,

$$\Pr[\mathsf{S}^{\mathrm{ke}}_5] = \Pr[\mathsf{S}^{\mathrm{ke}}_6] = 0.$$

Summing up the probabilities in the above games, we obtain the result of Lemma 2.

B.2 Proof of Theorem 4

The proof of Theorem 4 consists of the proof of Lemma 3 and the proof of Lemma 4.

B.2.1 Proof of Lemma 3

Let $\mathsf{S}_i^{\text{auth}}$ denote an event that there exists an authentication-adversary wins in Game i.

Game 0. This game equals the real security experiment described in Section 4.5. We have that

$$\Pr[\mathsf{S}_0^{\text{auth}}] = \epsilon_{\mathsf{auth}}.$$

Game 1. In this game, the challenger $\mathcal{C}$ proceeds exactly like the previous game, but aborts if two oracles generate the same randomness (i.e., either $\tilde{r}_1$ or $\tilde{r}_2$). Due to the birthday paradox, we have that

$$\Pr[\mathsf{S}_0^{\text{auth}}] \leq \Pr[\mathsf{S}_1^{\text{auth}}] + \frac{\rho^2}{2^{\ell_r}}.$$

So that each invocation of $\mathsf{TKE.MSG}$ takes as input a unique randomness.

Game 2. In this game, $\mathcal{C}$ proceeds as the previous game, but aborts if two oracles generate the same ephemeral public key of TKE (i.e., either m_1 or m_2). Let ϵ_{coll} note the event that two oracles have the identical ephemeral public keys. From [130, Lemma1], we have that if TKE is $(t, \epsilon_{\mathsf{TKE}})$-passively-secure without long-term key, then all ephemeral public keys generated by $\mathsf{TKE.MSG}$ in the runs of TKE are $(\rho, t, \epsilon_{coll})$-distinct such that $\epsilon_{coll} \leq \rho \cdot \epsilon_{\mathsf{TKE}}$. Since there are two honest parties and each party has ρ oracles, we have that

$$\Pr[\mathsf{S}_1^{\text{auth}}] \leq \Pr[\mathsf{S}_2^{\text{auth}}] + 2\rho \cdot \epsilon_{\mathsf{TKE}}.$$

As a result, each session identifier sid including $m_1||m_2$ is uniquely shared with its partner oracle.

Game 3. In this game, $\mathcal{C}$ guesses which the corruption case would occur (as in the proof of Lemma 1). $\mathcal{C}$ aborts if it fails in such a guess. The probability that $\mathcal{C}$ succeeds in guessing the corruption case is bounded by 1/3. Thus, we have that

$$\Pr[\mathsf{S}_2^{\text{auth}}] = 3 \cdot \Pr[\mathsf{S}_3^{\text{auth}}].$$

Game 4. Let S be the set of indices that are submitted to the HTLeak query. In this game, when the corruption case 2 or case 3 occurs, then $\mathcal{C}$ aborts if there is an oracle $\pi^u_{\mathcal{ID}^*}$ which accepts maliciously and sends out a selection set I^*_P such that $\mathsf{I}^*_P \subseteq \mathsf{S}$ (which means all secrets associated with indices in I^*_P are leaked), where $P^* \in \{C, S\}$. Note that I^*_P is chosen from an index set with size at least $L - z$. As in Game 3 in the proof of Lemma 1, We bound the probability

$$\Pr[\mathsf{I}^*_P \subseteq \mathsf{S}] = \frac{\binom{q_l}{z}}{\binom{L-z}{z}} < (\frac{q_l}{L-z})^z.$$

Since there are at most 2ρ such honest selection sets (for either prover or verifier) would be chosen, we have that

$$\Pr[\mathsf{S}_3^{\text{auth}}] < \Pr[\mathsf{S}_4^{\text{auth}}] + 2\rho \cdot (\frac{q_l}{L-z})^z.$$

Game 5. In this game, $\mathcal{C}$ aborts if one of the uncompromised secrets (which could be any factors) is asked by the adversary $\mathcal{A}$ in a random oracle query.

With the similar argument in Game 4 in the proof of Lemma 1, we have that

$$\Pr[\mathsf{S}_4^{\text{auth}}] = \Pr[\mathsf{S}_5^{\text{auth}}] + \frac{(3+L)q_h}{2^{\ell_p}}.$$

Game 6. $\mathcal{C}$ proceeds this game exactly as before, but aborts if an oracle $\pi^u_{\mathcal{ID}^*}$ such that $\mathsf{FullC}(\mathsf{id_C}^*, \mathsf{pid}^u_{\mathsf{id_C}^*}) = 0$ (throughout the game) received an $X^u_{\mathsf{id_C}^*}$ which is not sent from its partner oracle. We would like to bound the probability that an adversary's probability on forging $X^u_{\mathsf{id_C}^*}$. As stated in Game 5 in the proof of Lemma 1, we have that

$$\Pr[\mathsf{S}^{\text{auth}}_5] < \Pr[\mathsf{S}^{\text{auth}}_6] + \rho \cdot (\frac{q_l}{L})^z + \frac{\rho \cdot q_h}{2^{\ell_p}}.$$

Game 7. In this game, $\mathcal{C}$ aborts if $\mathcal{A}$ asked a random oracle query with the value $Y^u_{\mathcal{ID}^*}$ of an oracle $\pi^u_{\mathcal{ID}^*}$ such that $\mathsf{FullC}(\mathcal{ID}^*, \mathsf{pid}^u_{\mathcal{ID}^*}) = 0$ throughout the game. As $Y^u_{\mathcal{ID}^*}$ is computed involving a secret tag t_{i^*} which is not exposed. Furthermore, $Y^u_{\mathcal{ID}^*}$ is hidden by the hash function. Hence, $\mathcal{A}$ can only submit guessed $Y^u_{\mathcal{ID}^*}$ to random oracle queries. Since there are at most 2ρ honest oracles that an adversary may try to attack, we have that

$$\Pr[\mathsf{S}^{\text{auth}}_6] = \Pr[\mathsf{S}^{\text{auth}}_7] + \frac{2\rho \cdot q_h}{2^{\ell_p}}.$$

Game 8. In this game, for each oracle $\pi^u_{\mathcal{ID}^*}$ such that $\mathsf{FullC}(\mathcal{ID}^*, \mathsf{pid}^u_{\mathcal{ID}^*}, q_l) = 0$, $\pi^u_{\mathcal{ID}^*}$ rejects if it receives a message which is not sent by its partner oracle having a matching conversation to $\pi^u_{\mathcal{ID}^*}$. Since $\mathcal{A}$ cannot compute $Y^u_{\mathcal{ID}^*}$ used by $\pi^u_{\mathcal{ID}^*}$ for verification, it is unable to distinguish this game from the previous game. Thus the advantage of $\mathcal{A}$ in this game is zero.

Summing up the probabilities in all the above games, we have the result of Lemma 3.

B.2.2 Proof of Lemma 4

Let S_i^{ke} denote an event that there exists a session-key-adversary answers the session-key-challenge correctly in Game i. We show the proof of this lemma by the following games.

Game 0. This game equals the real security experiment described in Section 4.5. We have that

$$\Pr[\mathsf{S}_0^{\text{ke}}] - 1/2 = \epsilon_{\mathsf{skey}}.$$

Game 1. In this game, $\mathcal{C}$ aborts if the owner of the test oracle $\pi_{\mathcal{ID}^*}^u$ such that $\mathsf{FullC}(\mathcal{ID}^*, \mathsf{pid}_{\mathcal{ID}^*}^u, q_l) = 0$, and $\pi_{\mathcal{ID}^*}^u$ accepts without a partner oracle at $\mathsf{pid}_{\mathcal{ID}^*}^u$. Due to the authentication property of the protocol, we have that

$$\Pr[\mathsf{S}_0^{\text{ke}}] = \Pr[\mathsf{S}_1^{\text{ke}}] + \epsilon_{\mathsf{auth}}.$$

Hence, the test oracle must have a matching conversation at the server.

Game 2. This game proceeds exactly as the previous game but $\mathcal{C}$ aborts if it fails to guess the test oracle $\pi_{\mathcal{ID}^*}^u$ and its partner oracle $\pi_{\mathsf{pid}_{\mathcal{ID}^*}^u}^s$ such that they have matching conversations. Since there are 2 honest parties and ρ oracles for each party, the probability that $\mathcal{C}$ guesses correctly is at least $1/(2\rho)^2$. Thus we have that

$$\Pr[\mathsf{S}_1^{\text{ke}}] \leq 4\rho^2 \cdot \Pr[\mathsf{S}_2^{\text{ke}}].$$

Game 3. In this game, $\mathcal{C}$ replaces the key $k^{ke,*}$ of the test oracle $\pi_{\mathcal{ID}^*}^u$ and its partner oracle $\pi_{\mathsf{pid}_{\mathcal{ID}^*}^u}^s$ with the same random value $\widetilde{K^{ke,*}}$. Note that the TKE protocol instance executed between the test oracle and its partner oracle only allows for passive adversaries due to the change in the previous game. If there exists an adversary $\mathcal{A}$

which can distinguish this game from the previous game, then we use it to construct an algorithm $\mathcal{B}$ to break the passive security of TKE as follows. We assume that $\mathcal{B}$ interacts with the TKE challenger $\mathcal{C}_{\mathsf{TKE}}$ through $\mathsf{Execute}$ query. Meanwhile, $\mathcal{B}$ simulates the AKE challenger in this game for $\mathcal{A}$ as follows:

- Initially, $\mathcal{B}$ implements all honest oracles.
- Meanwhile, $\mathcal{B}$ generates the ephemeral key (i.e., m_1 or m_2) for each oracle $\pi^s_{\mathcal{ID}_i}$ using the ephemeral randomness of her own choice and answers all oracle queries honestly except for the test oracle and its partner oracle.
- As for the correctly guessed test oracle $\pi^s_{\mathsf{pid}^u_{\mathcal{ID}^*}}$ and its partner oracle $\pi^{t^*}_j$, $\mathcal{B}$ queries $\mathcal{C}_{\mathsf{TKE}}$ for executing a TKE test protocol instance and obtains (T^*, K^*_b) from $\mathcal{C}_{\mathsf{TKE}}$. Otherwise $\mathcal{B}$ simulates the ephemeral keys of $\pi^u_{\mathcal{ID}^*}$ and $\pi^s_{\mathsf{pid}^u_{\mathcal{ID}^*}}$ using the transcript T^*, and uses K^*_b to compute the session key of the test oracle.
- Eventually, $\mathcal{B}$ returns the bit b' given by $\mathcal{A}$ to $\mathcal{C}_{\mathsf{TKE}}$.

The simulation of $\mathcal{B}$ is perfect since $\mathcal{B}$ can always correctly answer all queries from $\mathcal{A}$. If $\mathcal{A}$ can correctly answer the bit b of the Test query with non-negligible probability, so can $\mathcal{B}$. By applying the security of TKE, we obtain that

$$\Pr[\mathsf{S}^{\mathrm{ke}}_2] \leq \Pr[\mathsf{S}^{\mathrm{ke}}_3] + \epsilon_{\mathsf{TKE}}.$$

Game 4. We replace the session key of the test oracle and its partner oracle (if any) with a truly random key that is independent of the bit chosen by the test oracle. This is possible since the test oracle would submit a random key material $\widetilde{K^{ke,*}}$ to the random oracle which results in a random session key as well. Thus the adversary gains no advantage in this game, i.e.,

$$\Pr[\mathsf{S}_3^{\mathrm{ke}}] = \Pr[\mathsf{S}_4^{\mathrm{ke}}] = 0.$$

Summing up the probabilities in the above games, we obtain the result of Lemma 4.

Bibliography

[1] S. Adepu and A. Mathur, "Distributed detection of single-stage multipoint cyber attacks in a water treatment plant," in *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security.* ACM, 2016, pp. 449–460.

[2] S. Agrawal, D. Boneh, X. Boyen, and D. M. Freeman, "Preventing pollution attacks in multi-source network coding," in *International Workshop on Public Key Cryptography.* Springer, 2010, pp. 161–176.

[3] C. M. Ahmed, M. Ochoa, J. Zhou, A. P. Mathur, R. Qadeer, C. Murguia, and J. Ruths, "Noiseprint: attack detection using sensor and process noise fingerprint in cyber physical systems," in *Proceedings of the 2018 on Asia Conference on Computer and Communications Security.* ACM, 2018, pp. 483–497.

[4] M. Ajtai, J. Komlós, and E. Szemerédi, "An o(n log n) sorting network," in *Proceedings of the fifteenth annual ACM symposium on Theory of computing.* ACM, 1983, pp. 1–9.

[5] M. A. Al Faruque, S. R. Chhetri, A. Canedo, and J. Wan, "Forensics of thermal side-channel in additive manufacturing systems," *University of California,*

Irvine, 2016.

[6] M. Allen, A. Prels, M. Lqbal, S. Srirangarajan, H. B. Llm, L. Glrod, and A. J. Whittle, "Real-time in-network distribution system monitoring to improve operational efficiency," *Journal-American Water Works Association*, vol. 103, no. 7, pp. 63–75, 2011.

[7] Allen-Bradley, "Micrologix 1400 programmable logic controller systems," [Accessed Jan., 2018]. [Online]. Available: https://goo.gl/9Xt3QJ

[8] R. Alur, *Principles of cyber-physical systems.* MIT Press, 2015.

[9] T. R. Alves, "Openplc: Getting started on raspberry pi," [Accessed Jan., 2018]. [Online]. Available: https://goo.gl/CgFXWz

[10] T. R. Alves, M. Buratto, F. M. de Souza, and T. V. Rodrigues, "Openplc: An open source alternative to automation," in *Global Humanitarian Technology Conference (GHTC), 2014 IEEE.* IEEE, 2014, pp. 585–589.

[11] L. Alvisi, A. Clement, A. Epasto, S. Lattanzi, and A. Panconesi, "Sok: The evolution of sybil defense via social networks," in *2013 ieee symposium on security and privacy.* IEEE, 2013, pp. 382–396.

[12] O. Andreeva, S. Gordeychik, G. Gritsai, O. Kochetova, E. Potseluevskaya, S. I. Sidorov, and A. A. Timorin, "Industrial control systems vulnerabilities statistics," 2016, [Accessed Jun., 2019]. [Online]. Available: https://media.kasperskycontenthub.com/wp-content/uploads/sites/43/2016/07/07190426/KL_REPORT_ICS_Statistic_vulnerabilities.pdf

[13] Apple, “Differential privacy overview,” [Accessed Jun., 2019]. [Online]. Available: https://images.apple.com/privacy/docs/Differential_Privacy_Overview.pdf

[14] Y. Aumann, Y. Z. Ding, and M. O. Rabin, “Everlasting security in the bounded storage model,” *IEEE Trans. Info. Theory*, vol. 48, no. 6, pp. 1668–1680, 2002.

[15] K. E. Batcher, “Sorting networks and their applications,” in *Proceedings of the April 30–May 2, 1968, spring joint computer conference.* ACM, 1968, pp. 307–314.

[16] D. Beaver, S. Micali, and P. Rogaway, “The round complexity of secure protocols,” in *STOC*, vol. 90, 1990, pp. 503–513.

[17] BECKHOFF, “Beckhoff new automation technology,” [Accessed Jan., 2018]. [Online]. Available: https://goo.gl/H3FcB7

[18] M. Bellare, V. T. Hoang, S. Keelveedhi, and P. Rogaway, “Efficient garbling from a fixed-key blockcipher,” in *2013 IEEE Symposium on Security and Privacy.* IEEE, 2013, pp. 478–492.

[19] M. Bellare, V. T. Hoang, and P. Rogaway, “Foundations of garbled circuits,” in *Proceedings of the 2012 ACM conference on Computer and communications security.* ACM, 2012, pp. 784–796.

[20] M. Bellare and S. K. Miner, “A forward-secure digital signature scheme,” in *Annual International Cryptology Conference.* Springer, 1999, pp. 431–448.

[21] M. Bellare and P. Rogaway, “Entity authentication and key distribution,” in *CRYPTO*, ser. LNCS, vol. 773. Springer, 1993, pp. 232–249.

[22] ——, "Random oracles are practical: A paradigm for designing efficient protocols," in *ACM CCS*. ACM, 1993, pp. 62–73.

[23] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the twentieth annual ACM symposium on Theory of computing*. ACM, 1988, pp. 1–10.

[24] C. Bing, "Trisis has the security world spooked, stumped and searching for answers," [Accessed Oct., 2018]. [Online]. Available: https://www.cyberscoop.com/trisis-ics-malware-saudi-arabia/

[25] A. Boldyreva, N. Chenette, Y. Lee, and A. O'neill, "Order-preserving symmetric encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2009, pp. 224–241.

[26] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *Annual Cryptology Conference*. Springer, 2011, pp. 578–595.

[27] W. Bolton, *Programmable logic controllers*. Newnes, 2015.

[28] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *CRYPTO*, ser. LNCS, vol. 10991. Springer, 2018, pp. 757–788.

[29] K. D. Bowers, C. Hart, A. Juels, and N. Triandopoulos, "Pillarbox: Combating next-generation malware with fast forward-secure logging," in *International Workshop on Recent Advances in Intrusion Detection*. Springer, 2014, pp. 46–67.

[30] R. R. Brooks and S. S. Iyengar, "Robust distributed computing and sensing algorithm," *Computer*, vol. 29, no. 6, pp. 53–60, 1996.

[31] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography," *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, 2005.

[32] A. Cardenas, S. Amin, B. Sinopoli, A. Giani, A. Perrig, S. Sastry *et al.*, "Challenges for securing cyber physical systems," in *Workshop on future directions in cyber-physical systems security*, vol. 5, no. 1, 2009.

[33] H. Carter, C. Lever, and P. Traynor, "Whitewash: Outsourcing garbled circuit generation for mobile devices," in *Proceedings of the 30th Annual Computer Security Applications Conference.* ACM, 2014, pp. 266–275.

[34] H. Carter, B. Mood, P. Traynor, and K. Butler, "Secure outsourced garbled circuit evaluation for mobile devices," *Journal of Computer Security*, vol. 24, no. 2, pp. 137–180, 2016.

[35] J. H. Castellanos, D. Antonioli, N. O. Tippenhauer, and M. Ochoa, "Legacy-compliant data authentication for industrial control system traffic," in *ACNS*. Springer, 2017, pp. 665–685.

[36] C. Castelluccia, A. C. Chan, E. Mykletun, and G. Tsudik, "Efficient and provably secure aggregation of encrypted data in wireless sensor networks," *ACM Transactions on Sensor Networks (TOSN)*, vol. 5, no. 3, p. 20, 2009.

[37] S. Challa, M. Wazid, A. K. Das, N. Kumar, G. R. Alavalapati, E. Yoon, and K. Yoo, "Secure signature-based authenticated key establishment scheme for future iot applications," *IEEE Access*, vol. 5, pp. 3028–3043, 2017.

[38] A. C. Chan, "Efficient defence against misbehaving TCP receiver dos attacks," *Computer Networks*, vol. 55, no. 17, pp. 3904–3914, 2011.

[39] A. C. Chan, J. W. Wong, J. Zhou, and J. C. M. Teo, "Scalable two-factor authentication using historical data," in *ESORICS*, ser. LNCS, vol. 9878. Springer, 2016, pp. 91–110.

[40] T.-H. H. Chan, E. Shi, and D. Song, "Privacy-preserving stream aggregation with fault tolerance," in *International Conference on Financial Cryptography and Data Security*. Springer, 2012, pp. 200–214.

[41] D. Chattaraj, M. Sarma, and A. K. Das, "A new two-server authentication and key agreement protocol for accessing secure cloud services," *Computer Networks*, vol. 131, pp. 144–164, 2018.

[42] B. Chen, C. Schmittner, Z. Ma, W. G. Temple, X. Dong, D. L. Jones, and W. H. Sanders, "Security analysis of urban railway systems: the need for a cyber-physical perspective," in *International Conference on Computer Safety, Reliability, and Security*. Springer, 2014, pp. 277–290.

[43] L. Chen and C. J. Mitchell, Eds., *Security Standardisation Research - First International Conference, SSR 2014, London, UK, December 16-17, 2014. Proceedings*, ser. Lecture Notes in Computer Science, vol. 8893. Springer, 2014.

[44] Y. Chen, L. López-Santidrián, J. Martínez, and P. Castillejo, "A lightweight privacy protection user authentication and key agreement scheme tailored for the internet of things environment: Lightpriauth," *J. Sensors*, vol. 2018, p. 7574238, 2018.

[45] P. Chew and K. Marzullo, "Masking failures of multidimensional sensors," in *[1991] Proceedings Tenth Symposium on Reliable Distributed Systems.* IEEE, 1991, pp. 32–41.

[46] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms.* MIT press, 2009.

[47] H. Corrigan-Gibbs and D. Boneh, "Prio: Private, robust, and scalable computation of aggregate statistics," in *14th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 17)*, 2017, pp. 259–282.

[48] K. A. R. Craig Trivelpiece and R. Campero, "Machine-to-machine and machine to cloud end-to-end authentication and security," October 2016, uS Patent 15/091,634.

[49] A. K. Das, S. Kumari, V. Odelu, X. Li, F. Wu, and X. Huang, "Provably secure user authentication and key agreement scheme for wireless sensor networks," *Security and Communication Networks*, vol. 9, no. 16, pp. 3670–3687, 2016.

[50] T. Dierks and E. Rescorla, "The transport layer security (tls) protocol version 1.2," Internet Engineering Task Force (IETF), Tech. Rep., 2008.

[51] Y. Dodis, S. Guo, and J. Katz, “Fixing cracks in the concrete: Random oracles with auxiliary input, revisited,” in *EUROCRYPT*, ser. LNCS, vol. 10211, 2017, pp. 473–495.

[52] J. R. Douceur, “The sybil attack,” in *International workshop on peer-to-peer systems.* Springer, 2002, pp. 251–260.

[53] B. Dowling, M. Fischlin, F. Günther, and D. Stebila, “A cryptographic analysis of the TLS 1.3 handshake protocol candidates,” in *ACM CCS.* ACM, 2015, pp. 1197–1210.

[54] A. Drozhzhin, “Black hat usa 2015: The full story of how that jeep was hacked,” 2015. [Online]. Available: https://www.kaspersky.com/blog/blackhat-jeep-cherokee-hack-explained/9493/

[55] A. Dua, N. Kumar, A. K. Das, and W. Susilo, “Secure message communication protocol among vehicles in smart city,” *IEEE Trans. Vehicular Technology*, vol. 67, no. 5, pp. 4359–4373, 2018.

[56] C. Dwork, “Differential privacy,” *Encyclopedia of Cryptography and Security*, pp. 338–340, 2011.

[57] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor, “Our data, ourselves: Privacy via distributed noise generation,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques.* Springer, 2006, pp. 486–503.

[58] S. Dziembowski, “Intrusion-resilience via the bounded-storage model,” in *TCC*, ser. LNCS, vol. 3876. Springer, 2006, pp. 207–224.

[59] T. Elahi, G. Danezis, and I. Goldberg, “Privex: Private collection of traffic statistics for anonymous communication networks,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security.* ACM, 2014, pp. 1068–1079.

[60] Z. Erkin, A. Piva, S. Katzenbeisser, R. L. Lagendijk, J. Shokrollahi, G. Neven, and M. Barni, “Protection and retrieval of encrypted multimedia content: When cryptography meets signal processing,” *EURASIP Journal on Information Security*, vol. 2007, p. 17, 2007.

[61] Z. Erkin, J. R. Troncoso-Pastoriza, R. L. Lagendijk, and F. Pérez-González, “Privacy-preserving data aggregation in smart metering systems: An overview,” *IEEE Signal Processing Magazine*, vol. 30, no. 2, pp. 75–86, 2013.

[62] Z. Erkin and G. Tsudik, “Private computation of spatial and temporal power consumption with smart meters,” in *International Conference on Applied Cryptography and Network Security.* Springer, 2012, pp. 561–577.

[63] Ú. Erlingsson, V. Pihur, and A. Korolova, “Rappor: Randomized aggregatable privacy-preserving ordinal response,” in *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security.* ACM, 2014, pp. 1054–1067.

[64] A. Esfahani, G. Mantas, R. Matischek, F. B. Saghezchi, J. Rodriguez, A. Bicaku, S. Maksuti, M. G. Tauber, C. Schmittner, and J. Bastos, “A lightweight authentication mechanism for M2M communications in industrial iot environment,” *IEEE Internet of Things Journal*, vol. 6, no. 1, pp. 288–296, 2019.

[65] N. Falliere, L. O. Murchu, and E. Chien, "W32. stuxnet dossier," *White paper, Symantec Corp., Security Response*, vol. 5, no. 6, p. 29, 2011.

[66] A. Faruque, M. Abdullah, S. R. Chhetri, A. Canedo, and J. Wan, "Acoustic side-channel attacks on additive manufacturing systems," in *Proceedings of the 7th International Conference on Cyber-Physical Systems.* IEEE Press, 2016, p. 19.

[67] U. Feige, J. Kilian, and M. Naor, "A minimal model for secure computation," in *STOC*, vol. 94. Citeseer, 1994, pp. 554–563.

[68] J. Feigenbaum, B. Pinkas, R. Ryger, and F. Saint-Jean, "Secure computation of surveys," in *EU Workshop on Secure Multiparty Protocols*, 2004, pp. 2–14.

[69] N. Fleischhacker, M. Manulis, and A. Azodi, "A modular framework for multi-factor authentication and key exchange," in *Security Standardisation Research - First International Conference, SSR 2014, London, UK, December 16-17, 2014. Proceedings*, 2014, pp. 190–214.

[70] L. Garcia, F. Brasser, M. H. Cintuglu, A.-R. Sadeghi, O. A. Mohammed, and S. A. Zonouz, "Hey, my malware knows physics! attacking plcs with physical model aware rootkit." in *NDSS*, 2017.

[71] J. Goh, S. Adepu, K. N. Junejo, and A. Mathur, "A dataset to support research in the design of secure water treatment systems," in *International Conference on Critical Information Infrastructures Security.* Springer, 2016, pp. 88–99.

[72] O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game," in *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. ACM, 1987, pp. 218–229.

[73] N. Govil, A. Agrawal, and N. O. Tippenhauer, "On ladder logic bombs in industrial control systems," in *Computer Security*. Springer, 2017, pp. 110–126.

[74] V. Goyal, P. Mohassel, and A. Smith, "Efficient two party and multi party computation against covert adversaries," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2008, pp. 289–306.

[75] X. Guo, C. Jin, C. Zhang, A. Papadimitriou, D. Hély, and R. Karri, "Can algorithm diversity in stream cipher implementation thwart (natural and) malicious faults?" *IEEE Transactions on Emerging Topics in Computing*, vol. 4, no. 3, pp. 363–373, 2016.

[76] X. Guo, N. Karimi, F. Regazzoni, C. Jin, and R. Karri, "Simulation and analysis of negative-bias temperature instability aging on power analysis attacks," in *2015 IEEE International Symposium on Hardware Oriented Security and Trust*. IEEE, 2015, pp. 124–129.

[77] X. Guo, D. Mukhopadhyay, C. Jin, and R. Karri, "Nrepo: Normal basis recomputing with permuted operands," in *IEEE International Symposium on Hardware-Oriented Security and Trust*. IEEE, 2014, pp. 118–123.

[78] ——, “Security analysis of concurrent error detection against differential fault analysis,” *Journal of Cryptographic Engineering*, vol. 5, no. 3, pp. 153–169, 2015.

[79] S. K. Haider, C. Jin, M. Ahmad, D. Shila, O. Khan, and M. van Dijk, “Advancing the state-of-the-art in hardware trojans detection,” *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 1, pp. 18–32, 2019.

[80] S. K. Haider, C. Jin, and M. van Dijk, “Advancing the state-of-the-art in hardware trojans design,” in *2017 IEEE 60th International Midwest Symposium on Circuits and Systems*. IEEE, 2017, pp. 823–826.

[81] H. Handschuh, P. Paillier, and J. Stern, “Probing attacks on tamper-resistant devices,” in *International Workshop on Cryptographic Hardware and Embedded Systems*. Springer, 1999, pp. 303–315.

[82] J. He, Z. Yang, J. Zhang, W. Liu, and C. Liu, “On the security of a provably secure, efficient, and flexible authentication scheme for ad hoc wireless sensor networks,” *I. J. of Distributed Sensor Networks*, vol. 14, no. 1, pp. 1–11, 2018.

[83] R. B. Hee, “Knowing the basics of plcs,” [Accessed Jan., 2018]. [Online]. Available: https://goo.gl/S7DzB9

[84] Y. Huang, D. Evans, and J. Katz, “Private set intersection: Are garbled circuits better than custom protocols?” in *NDSS*, 2012.

[85] Y. Huang, D. Evans, J. Katz, and L. Malka, “Faster secure two-party computation using garbled circuits.” in *USENIX Security Symposium*, vol. 201, no. 1, 2011, pp. 331–335.

[86] I. E. C. (IEC), “Iec 61131-3:2013 programmable controllers - part 3: Programming languages,” 2013. [Online]. Available: https://webstore.iec.ch/publication/4552

[87] ——, “Iot 2020: Smart and secure iot platform,” 2016. [Online]. Available: https://www.iec.ch/whitepaper/iotplatform/

[88] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, “Generic compilers for authenticated key exchange,” in *ASIACRYPT*, ser. LNCS, vol. 6477. Springer, 2010, pp. 232–249.

[89] ——, “On the security of TLS-DHE in the standard model,” in *CRYPTO*, ser. LNCS, vol. 7417. Springer, 2012, pp. 273–293.

[90] S. Jarecki, H. Krawczyk, M. Shirvanian, and N. Saxena, “Two-factor authentication with end-to-end password security,” in *IACR International Workshop on Public Key Cryptography*. Springer, 2018, pp. 431–461.

[91] M. Jawurek and F. Kerschbaum, “Fault-tolerant privacy-preserving statistics,” in *International Symposium on Privacy Enhancing Technologies Symposium*. Springer, 2012, pp. 221–238.

[92] X. Jia, D. He, L. Li, and K. R. Choo, “Signature-based three-factor authenticated key exchange for internet of things applications,” *Multimed. Tools. Appl.*, vol. 77, no. 14, pp. 18 355–18 382, 2018.

[93] C. Jin, C. Herder, L. Ren, P. H. Nguyen, B. Fuller, S. Devadas, and M. van Dijk, “Fpga implementation of a cryptographically-secure puf based on learning parity with noise,” *Cryptography*, vol. 1, no. 3, p. 23, 2017.

[94] C. Jin, L. Ren, X. Liu, P. Zhang, and M. van Dijk, "Mitigating synchronized hardware trojan attacks in smart grids," in *Proceedings of the 2nd Workshop on Cyber-Physical Security and Resilience in Smart Grids*. ACM, 2017, pp. 35–40.

[95] C. Jin, S. Valizadeh, and M. van Dijk, "Snapshotter: Lightweight intrusion detection and prevention system for industrial control systems," in *2018 IEEE Industrial Cyber-Physical Systems*. IEEE, 2018, pp. 824–829.

[96] C. Jin and M. van Dijk, "Secure and efficient initialization and authentication protocols for shield," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 1, pp. 156–173, 2019.

[97] C. Jin, M. van Dijk, M. Reiter, and H. Zhang, "Pwop: Intrusion-tolerant and privacy-preserving sensor fusion," *IACR Cryptology ePrint Archive*, vol. 2018, p. 1171, 2018.

[98] C. Jin, Z. Yang, S. Adepu, and J. Zhou, "Hmake: Legacy-compliant multi-factor authenticated key exchange from historical data," *IACR Cryptology ePrint Archive*, vol. 2019, p. 450, 2019.

[99] B. Johnson, D. Caban, M. Krotofil, D. Scali, N. Brubaker, and C. Glyer, "Attackers deploy new ics attack framework "triton" and cause operational disruption to critical infrastructure," [Accessed Oct., 2018]. [Online]. Available: https://www.fireeye.com/blog/threat-research/2017/12/attackers-deploy-new-ics-attack-framework-triton.html

[100] M. Joye and B. Libert, "A scalable scheme for privacy-preserving aggregation of time-series data," in *International Conference on Financial Cryptography and Data Security.* Springer, 2013, pp. 111–125.

[101] S. Kamara, P. Mohassel, and M. Raykova, "Outsourcing multi-party computation." *IACR Cryptology ePrint Archive*, vol. 2011, p. 272, 2011.

[102] S. Kamara, P. Mohassel, M. Raykova, and S. Sadeghian, "Scaling private set intersection to billion-element sets," in *International Conference on Financial Cryptography and Data Security.* Springer, 2014, pp. 195–215.

[103] S. Kamara, P. Mohassel, and B. Riva, "Salus: a system for server-aided secure function evaluation," in *Proceedings of the 2012 ACM conference on Computer and communications security.* ACM, 2012, pp. 797–808.

[104] S. Karnouskos, "Stuxnet worm impact on industrial cyber-physical system security," in *IECON 2011-37th Annual Conference on IEEE Industrial Electronics Society.* IEEE, 2011, pp. 4490–4494.

[105] J. Katz and Y. Lindell, *Introduction to modern cryptography.* CRC press, 2014.

[106] G. S. Kc, A. D. Keromytis, and V. Prevelakis, "Countering code-injection attacks with instruction-set randomization," in *Proceedings of the 10th ACM conference on Computer and communications security.* ACM, 2003, pp. 272–280.

[107] A. Keliris and M. Maniatakos, "Icsref: A framework for automated reverse engineering of industrial control systems binaries," *arXiv preprint arXiv:1812.03478*, 2018.

[108] F. Kerschbaum and A. Schroepfer, “Optimal average-complexity ideal-security order-preserving encryption,” in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 275–286.

[109] R. S. Khan, N. Noor, C. Jin, S. Muneer, F. Dirisaglik, A. Ciardullo, P. H. Nguyen, M. van Dijk, A. Gokirmak, and H. Silva, “Exploiting lithography limits for hardware security applications,” in *2019 IEEE Conference on Nanotechnology*. IEEE, 2019.

[110] R. S. Khan, N. Noor, C. Jin, J. Scoggin, Z. Woods, S. Muneer, A. Ciardullo, P. H. Nguyen, A. Gokirmak, M. van Dijk, and H. Silva, “Phase change memory and its applications in hardware security,” in *Security Opportunities in Nano Devices and Emerging Technologies*. CRC Press, 2017, pp. 115–136.

[111] J. Klick, S. Lau, D. Marzin, J.-O. Malchow, and V. Roth, “Internet-facing plcs as a network backdoor,” in *Communications and Network Security (CNS), 2015 IEEE Conference on*. IEEE, 2015, pp. 524–532.

[112] E. D. Knapp and J. T. Langill, *Industrial Network Security: Securing critical infrastructure networks for smart grid, SCADA, and other Industrial Control Systems*. Syngress, 2014.

[113] E. D. Knapp and R. Samani, *Applied cyber security and the smart grid: implementing security controls into the modern power infrastructure*. Newnes, 2013.

[114] D. E. Knuth, *The art of computer programming: sorting and searching*. Pearson Education, 1997, vol. 3.

[115] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Annual International Cryptology Conference.* Springer, 1999, pp. 388–397.

[116] P. C. Kocher, "Timing attacks on implementations of diffie-hellman, rsa, dss, and other systems," in *Annual International Cryptology Conference.* Springer, 1996, pp. 104–113.

[117] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider, "Improved garbled circuit building blocks and applications to auctions and computing minima," in *International Conference on Cryptology and Network Security.* Springer, 2009, pp. 1–20.

[118] V. Kolesnikov and T. Schneider, "Improved garbled circuit: Free xor gates and applications," in *International Colloquium on Automata, Languages, and Programming.* Springer, 2008, pp. 486–498.

[119] D. Kravets, "Feds: Hacker disabled offshore oil platforms' leak-detection system," [Accessed Oct., 2018]. [Online]. Available: https://www.wired.com/2009/03/feds-hacker-dis/

[120] B. Kreuter, A. Shelat, B. Mood, and K. Butler, "{PCF}: A portable circuit format for scalable two-party secure computation," in *Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13)*, 2013, pp. 321–336.

[121] K. Kursawe, G. Danezis, and M. Kohlweiss, "Privacy-friendly aggregation for the smart-grid," in *International Symposium on Privacy Enhancing Technologies Symposium.* Springer, 2011, pp. 175–191.

[122] D. Kushner, "The real story of stuxnet," *IEEE Spectrum*, vol. 50, no. 3, pp. 48–53, 2013.

[123] C. Lamb, "Man pleads guilty in caliso computer attack," [Accessed Oct., 2018]. [Online]. Available: https://www.bizjournals.com/sacramento/stories/2007/12/10/daily65.html

[124] L. Lamport, "Using time instead of timeout for fault-tolerant distributed systems." *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 6, no. 2, pp. 254–280, 1984.

[125] ——, *Synchronizing time servers.* Digital, Systems Research Center, 1987.

[126] R. Langner, "Stuxnet: Dissecting a cyberwarfare weapon," *IEEE Security & Privacy*, vol. 9, no. 3, pp. 49–51, 2011.

[127] N. Leali, "Lessons from an insider attack on scada systems," Aug. 2009, [Accessed May., 2019]. [Online]. Available: https://blogs.cisco.com/security/lessons_from_an_insider_attack_on_scada_systems

[128] R. Lee, J. Slowik, B. Miller, A. Cherepanov, and R. Lipovsky, "Industroyer/crashoverride: Zero things cool about a threat group targeting the power grid," *Black Hat USA*, 2017.

[129] D. Li, H. Guo, J. Zhou, L. Zhou, and J. W. Wong, "Scadawall: A cpi-enabled firewall model for scada security," *Computers & Security*, vol. 80, pp. 134–154, 2019.

[130] Y. Li, S. Schäge, Z. Yang, C. Bader, and J. Schwenk, "New modular compilers for authenticated key exchange," in *ACNS*, ser. LNCS, vol. 8479. Springer, 2014, pp. 1–18.

[131] Y. Li, S. Schäge, Z. Yang, F. Kohlar, and J. Schwenk, "On the security of the pre-shared key ciphersuites of TLS," in *PKC*, ser. LNCS, vol. 8383. Springer, 2014, pp. 669–684.

[132] Y. Lindell and B. Pinkas, "A proof of security of yao's protocol for two-party computation," *Journal of cryptology*, vol. 22, no. 2, pp. 161–188, 2009.

[133] M. A. Lisovich, D. K. Mulligan, and S. B. Wicker, "Inferring personal information from demand-response systems," *IEEE Security & Privacy*, vol. 8, no. 1, pp. 11–20, 2010.

[134] C. Liu, X. S. Wang, K. Nayak, Y. Huang, and E. Shi, "Oblivm: A programming framework for secure computation," in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 359–376.

[135] C. Liu, P. Cronin, and C. Yang, "Securing cyber-physical systems from hardware trojan collusion," *IEEE Transactions on Emerging Topics in Computing*, 2017.

[136] Y. Liu, P. Ning, and M. K. Reiter, "False data injection attacks against state estimation in electric power grids," *ACM Transactions on Information and System Security (TISSEC)*, vol. 14, no. 1, p. 13, 2011.

[137] L. Ljung, "System identification," *Wiley Encyclopedia of Electrical and Electronics Engineering*, pp. 1–19, 1999.

[138] D. Ma and G. Tsudik, "A new approach to secure logging," *ACM Transactions on Storage (TOS)*, vol. 5, no. 1, p. 2, 2009.

[139] H. Maleki, R. Rahaeimehr, C. Jin, and M. van Dijk, "New clone-detection approach for rfid-based supply chains," in *IEEE International Symposium on Hardware Oriented Security and Trust*. IEEE, 2017, pp. 122–127.

[140] H. Maleki, S. Valizadeh, W. Koch, A. Bestavros, and M. van Dijk, "Markov modeling of moving target defense games," in *Proceedings of the 2016 ACM Workshop on Moving Target Defense*. ACM, 2016, pp. 81–92.

[141] D. Malkhi, N. Nisan, B. Pinkas, Y. Sella *et al.*, "Fairplay-secure two-party computation system." in *USENIX Security Symposium*, vol. 4. San Diego, CA, USA, 2004, p. 9.

[142] K. Marzullo, "Tolerating failures of continuous-valued sensors," *ACM Transactions on Computer Systems (TOCS)*, vol. 8, no. 4, pp. 284–304, 1990.

[143] K. Marzullo and S. Owicki, "Maintaining the time in a distributed system," in *Proceedings of the second annual ACM symposium on Principles of distributed computing*. ACM, 1983, pp. 295–305.

[144] K. A. Marzullo, "Maintaining the time in a distributed system: an example of a loosely-coupled distributed service (synchronization, fault-tolerance, debugging)," 1984.

[145] A. Matrosov, E. Rodionov, D. Harley, and J. Malcho, "Stuxnet under the microscope," *ESET LLC (September 2010)*, 2010.

[146] P. McDaniel and S. McLaughlin, "Security and privacy challenges in the smart grid," *IEEE Security & Privacy*, vol. 7, no. 3, pp. 75–77, 2009.

[147] S. McLaughlin, C. Konstantinou, X. Wang, L. Davi, A.-R. Sadeghi, M. Maniatakos, and R. Karri, "The cybersecurity landscape in industrial control systems," *Proceedings of the IEEE*, vol. 104, no. 5, pp. 1039–1057, 2016.

[148] R. McMillan, "Hackers break into water system network," https://www.computerworld.com/article/2547938/security0/hackers-break-into-water-system-network.html, [Accessed Oct., 2018.

[149] ——, "Insider charged with hacking california canal system," [Accessed Oct., 2018]. [Online]. Available: https://www.computerworld.com/article/2540235/disaster-recovery/insider-charged-with-hacking-california-canal-system.html

[150] L. Melis, G. Danezis, and E. De Cristofaro, "Efficient private statistics with succinct sketches," *arXiv preprint arXiv:1508.06110*, 2015.

[151] S. A. Milinković and L. R. Lazić, "Industrial plc security issues," in *2012 20th Telecommunications Forum (TELFOR)*. IEEE, 2012, pp. 1536–1539.

[152] "Miracl cryptographic library," 2018. [Online]. Available: https://bit.ly/2MltKVG

[153] B. Mood, D. Gupta, K. Butler, and J. Feigenbaum, "Reuse it or lose it: More efficient secure computation through reuse of encrypted values," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2014, pp. 582–596.

[154] M. Naor, B. Pinkas, and R. Sumner, “Privacy preserving auctions and mechanism design,” *EC*, vol. 99, pp. 129–139, 1999.

[155] B. News, “Colombia rebels blast power pylons,” [Accessed Oct., 2018]. [Online]. Available: http://news.bbc.co.uk/2/hi/americas/607782.stm

[156] P. H. Nguyen, D. P. Sahoo, C. Jin, K. Mahmood, U. Rührmair, and M. van Dijk, “The interpose puf: Secure puf design against state-of-the-art machine learning attacks,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, 2019.

[157] J. B. Nielsen and C. Orlandi, “Lego for two-party secure computation,” in *Theory of Cryptography Conference.* Springer, 2009, pp. 368–386.

[158] L. P. Urien and P. Martin, “EMV support for TLS-PSK,” draft-urien-tls-psk-emv-02, TLS Working Group, Feb. 2011. [Online]. Available: http://tools.ietf.org/html/draft-urien-tls-psk-emv-02

[159] R. A. Popa, F. H. Li, and N. Zeldovich, “An ideal-security protocol for order-preserving encoding,” in *2013 IEEE Symposium on Security and Privacy.* IEEE, 2013, pp. 463–477.

[160] B. Przydatek, D. Song, and A. Perrig, “Sia: Secure information aggregation in sensor networks,” in *Proceedings of the 1st international conference on Embedded networked sensor systems.* ACM, 2003, pp. 255–265.

[161] N. F. Pub, “197: Advanced encryption standard (aes),” *Federal information processing standards publication*, vol. 197, no. 441, p. 0311, 2001.

[162] D. Quarta, M. Pogliani, M. Polino, F. Maggi, A. M. Zanchettin, and S. Zanero, "Breaking the laws of robotics attacking industrial robots," 2017. [Online]. Available: https://www.blackhat.com/docs/us-17/thursday/us-17-Quarta-Breaking-The-Laws-Of-Robotics-Attacking-Industrial-Robots.pdf

[163] E. Rescorla, "The transport layer security (tls) protocol version 1.3," Internet Engineering Task Force (IETF), Tech. Rep., 2018.

[164] A. Rial and G. Danezis, "Privacy-preserving smart metering," in *Proceedings of the 10th annual ACM workshop on Privacy in the electronic society*. ACM, 2011, pp. 49–60.

[165] J. E. Savage, *Models of computation*. Addison-Wesley Reading, MA, 1998, vol. 136.

[166] K. Scarfone and P. Mell, "Guide to intrusion detection and prevention systems (idps)," *NIST special publication*, vol. 800, no. 2007, p. 94, 2007.

[167] U. Schmid and K. Schossmaier, "How to reconcile fault-tolerant interval intersection with the lipschitz condition," *Distributed Computing*, vol. 14, no. 2, pp. 101–111, 2001.

[168] B. Schneier and J. Kelsey, "Cryptographic support for secure logs on untrusted machines." in *USENIX Security Symposium*, vol. 98, 1998, pp. 53–62.

[169] T. Seals, "Sas 2019: Triton ics malware hits a second victim," 2019, [Accessed Jun., 2019]. [Online]. Available: https://threatpost.com/triton-ics-malware-second-victim/143658/

[170] H. Shacham and B. Waters, “Compact proofs of retrievability,” *J. Cryptology*, vol. 26, no. 3, pp. 442–483, 2013.

[171] C.-h. Shen *et al.*, “Two-output secure computation with malicious adversaries,” in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2011, pp. 386–405.

[172] E. Shi, H. Chan, E. Rieffel, R. Chow, and D. Song, “Privacy-preserving aggregation of time-series data,” in *Annual Network & Distributed System Security Symposium (NDSS)*. Internet Society., 2011.

[173] SIEMENS, “Simatic et 200s im 151-7 cpu interface module manuel,” [Accessed Jan., 2018]. [Online]. Available: https://goo.gl/itPEBe

[174] ——, “Simatic process historian,” https://w3.siemens.com/mcms/automation-software/en/scada-software/scada-options/simatic-process-historian/pages/default.aspx.

[175] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, “Industrial internet of things: Challenges, opportunities, and directions,” *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[176] J. Slay and M. Miller, “Lessons learned from the maroochy water breach,” in *International Conference on Critical Infrastructure Protection*. Springer, 2007, pp. 73–82.

[177] E. M. Songhori, S. U. Hussain, A.-R. Sadeghi, T. Schneider, and F. Koushanfar, “Tinygarble: Highly compressed and scalable sequential garbled circuits,” in *2015 IEEE Symposium on Security and Privacy*. IEEE, 2015, pp. 411–428.

[178] J. Staggs, "Adventures in attacking wind farm control networks," *Black Hat USA*, 2017.

[179] S. H. Standard, "Fips pub 180-2," *National Institute of Standards and Technology*, 2002.

[180] K. Stouffer, J. Falco, and K. Scarfone, "Guide to industrial control systems (ics) security," *NIST special publication*, vol. 800, no. 82, pp. 16–16, 2011.

[181] Symantec Security Response, "Triton: New Malware Threatens Industrial Safety Systems," Dec. 2017, [Accessed Jan., 2018]. [Online]. Available: https://goo.gl/66Mvt7

[182] M. Tehranipoor, D. Forte, G. S. Rose, and S. Bhunia, *Security Opportunities in Nano Devices and Emerging Technologies.* CRC Press, 2017.

[183] F. Tramarin, A. K. Mok, and S. Han, "Real-time and reliable industrial control over wireless lans: Algorithms, protocols, and future directions," *Proceedings of the IEEE*, vol. 107, no. 6, pp. 1027–1052, 2019.

[184] N. Tuptuk and S. Hailes, "Security of smart manufacturing systems," *Journal of manufacturing systems*, vol. 47, pp. 93–106, 2018.

[185] M. van Dijk, C. Jin, H. Maleki, P. H. Nguyen, and R. Rahaeimehr, "Weak-unforgeable tags for secure supply chain management," in *2018 International Conference on Financial Cryptography and Data Security*, 2018.

[186] J. Voas, "Networks of 'things'," *NIST Special Publication*, vol. 800, no. 183, pp. 800–183, 2016.

[187] G. Wang, T. Luo, M. T. Goodrich, W. Du, and Z. Zhu, "Bureaucratic protocols for secure two-party sorting, selection, and permuting," in *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*. ACM, 2010, pp. 226–237.

[188] M. Wazid, A. K. Das, V. Odelu, N. Kumar, M. Conti, and M. Jo, "Design of secure user authenticated key management protocol for generic iot networks," *IEEE Internet of Things Journal*, vol. 5, no. 1, pp. 269–282, 2018.

[189] D. Westhoff, J. Girao, and M. Acharya, "Concealed data aggregation for reverse multicast traffic in sensor networks: Encryption, key distribution, and routing adaptation," *IEEE Transactions on Mobile Computing*, vol. 5, no. 10, pp. 1417–1431, 2006.

[190] X. Xing, W. Meng, D. Doozan, A. C. Snoeren, N. Feamster, and W. Lee, "Take this personally: Pollution attacks on personalized services," in *Presented as part of the 22nd {USENIX} Security Symposium ({USENIX} Security 13)*, 2013, pp. 671–686.

[191] W. Yan, C. Jin, F. Tehranipoor, and J. A. Chandy, "Phase calibrated ring oscillator puf design and implementation on fpgas," in *International Conference on Field Programmable Logic and Applications*. IEEE, 2017, pp. 1–8.

[192] Z. Yang and J. Lai, "New constructions for (multiparty) one-round key exchange with strong security," *SCIENCE CHINA Information Sciences*, vol. 61, no. 5, pp. 059 102:1–059 102:3, 2018.

[193] Z. Yang, J. Lai, C. Liu, W. Liu, and S. Li, "Simpler generic constructions for strongly secure one-round key exchange from weaker assumptions," *Comput. J.*, vol. 60, no. 8, pp. 1145–1160, 2017.

[194] Z. Yang, J. Lai, Y. Sun, and J. Zhou, "A novel authenticated key agreement protocol with dynamic credential for wsns," *ACM Trans. Sen. Netw.*, vol. 15, no. 2, March 2019. [Online]. Available: https://doi.org/10.1145/3303704

[195] Z. Yang, C. Liu, W. Liu, S. Luo, H. Long, and S. Li, "A lightweight generic compiler for authenticated key exchange from non-interactive key exchange with auxiliary input," *I. J. Network Security*, vol. 18, no. 6, pp. 1109–1121, 2016.

[196] A. C.-C. Yao, "How to generate and exchange secrets," in *27th Annual Symposium on Foundations of Computer Science (sfcs 1986).* IEEE, 1986, pp. 162–167.

[197] A. A. Yavuz, P. Ning, and M. K. Reiter, "Efficient, compromise resilient and append-only cryptographic schemes for secure audit logging," in *International Conference on Financial Cryptography and Data Security.* Springer, 2012, pp. 148–163.

[198] A. A. Yavuz and P. Ning, "Baf: An efficient publicly verifiable secure audit logging scheme for distributed systems," in *Computer Security Applications Conference, 2009. ACSAC'09. Annual.* IEEE, 2009, pp. 219–228.

[199] N. Youdao, "P4p: practical large-scale privacy-preserving distributed computation robust against malicious users," *In Proc USENEX*, 2010.

[200] S. Zahur and D. Evans, "Obliv-c: A language for extensible data-oblivious computation." *IACR Cryptology ePrint Archive*, vol. 2015, p. 1153, 2015.

[201] S. Zahur, M. Rosulek, and D. Evans, "Two halves make a whole," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 220–250.

[202] R. Zhang, Y. Xiao, S. Sun, and H. Ma, "Efficient multi-factor authenticated key exchange scheme for mobile communications," *IEEE Transactions on Dependable and Secure Computing*, 2017.