Zero-Knowledge Realization of Software-Defined Gateway in Fog Computing

Te-Yuan Lin[1] and Chiou-Shann Fuh[2]
[1] Department of Computer Science and Information Engineering, National Taiwan University
Taipei, Taiwan - ROC
[e-mail: d03922002@ntu.edu.tw]
[2] Department of Computer Science and Information Engineering, National Taiwan University
Taipei, Taiwan - ROC
[e-mail: fuh@csie.ntu.edu.tw]
*Corresponding author: Te-Yuan Lin

Received May 9, 2018; revised July 6, 2018; accepted August 15, 2018; published December 31, 2018

Abstract

Driven by security and real-time demands of Internet of Things (IoT), the timing of fog computing and edge computing have gradually come into place. Gateways bear more nearby computing, storage, analysis and as an intelligent broker of the whole computing lifecycle in between local devices and the remote cloud. In fog computing, the edge broker requires X-aware capabilities that combines software programmability, stream processing, hardware optimization and various connectivity to deal with such as security, data abstraction, network latency, service classification and workload allocation strategy. The prosperous of Field Programmable Gate Array (FPGA) pushes the possibility of gateway capabilities further landed. In this paper, we propose a software-defined gateway (SDG) scheme for fog computing paradigm termed as Fog Computing Zero-Knowledge Gateway that strengthens data protection and resilience merits designed for industrial internet of things or highly privacy concerned hybrid cloud scenarios. It is a proxy for fog nodes and able to integrate with existing commodity gateways. The contribution is that it converts Privacy-Enhancing Technologies rules into provable statements without knowing original sensitive data and guarantees privacy rules applied to the sensitive data before being propagated while preventing potential leakage threats. Some logical functions can be offloaded to any programmable micro-controller embedded to achieve higher computing efficiency.

Keywords: Cloud security, Fog Computing, IoT, SDG, Zero-Knowledge

1. Introduction

The terms fog computing and edge computing are used interchangeably often in the industry, yet they have differences in emphasized nature. Both are pushing intelligence and processing capabilities to where the data is collected to minimize latency, however, fog computing pays more attention on applications of fog nodes and routes filtered data to the optimal place for analysis while sustaining the data privacy inside the protected network. Fog nodes can be equipped with quite powerful computing, scalable memory or storage capacity and inter-connected in a mesh network topology to form as a fog layer infrastructure. The fog layer plays as a bridge as illustrated in **Fig. 1**[1][2], it is closer to data source than cloud and hence to achieve lower transmission latency, to strengthen managed capabilities and to orchestrate the heterogeneous application and devices in between the cloud layer and the thing layer. For industrial scenarios, realizing resilient connectivity and intelligent computing possibility are just right characteristics in favor for fog computing yet hard to gain from public cloud platform.

Though the concept of fog computing is emerging prosperously, more detail of applied problems need to be rethought on fog layer [1]. For instances, the first challenge is how to deal with nodes faults and updates. A failed node cannot be capable to function its role while a rogue node could pretend to be legitimate to exchange and collect the data generated by other Internet of Things (IoT) devices for malicious purposes. The second is security and privacy concerns. At first glance, security and privacy are more affirmed in fog than in pure cloud world owing to everything is not transmitted out to cloud yet before the pre-process done by fog layer, however, how do we confirm the required pre-processes are all applied? Any existing man-in-the-middle threats? If we turn around heads to the other end — sensors, actuators and devices of the thing layer, they are usually quite severely resources-constrained with intermittently connectivity. Many existing securing protocols are built on time synchronization over wireless packet transmissions or resource-intensive authentication processing and they are not efficiently suitable for resource-constrained IoT devices, so it is rather consolidating data centrally than letting them connect to internet directly, for safety's sake.

The above requires more intelligent abilities to take local decisions by fog layer to bridge the cloud and the thing, that is the reason why we consider software-defined capabilities [2] of fog layer is worth further research in terms of the key accelerator to widespread IoT Industry 4.0.

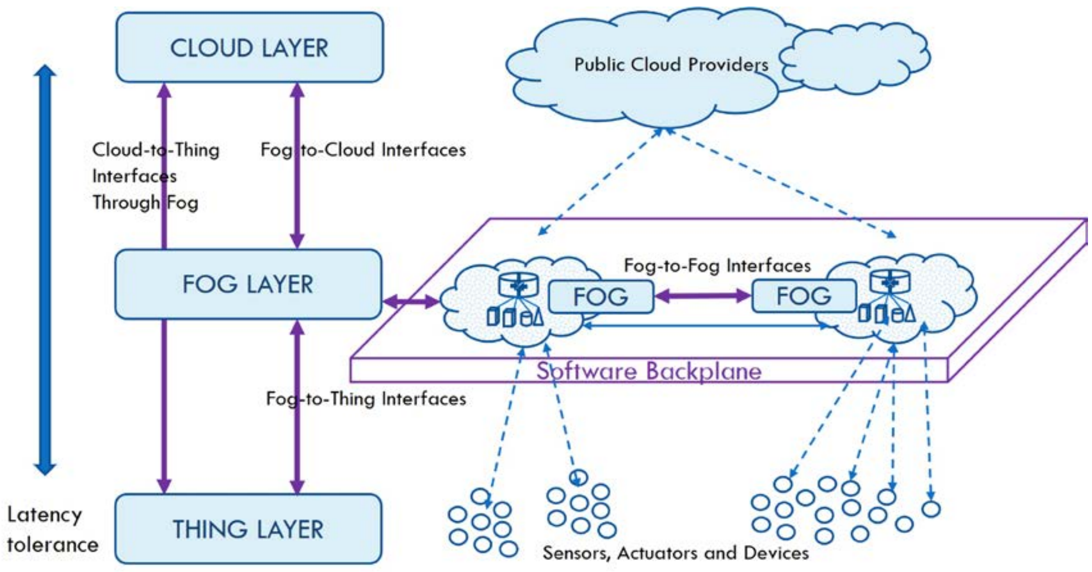


Fig. 1. Fog computing hierarchical and interfaces for industrial IoT

2. Challenges and Related Work

Diving into a number of local decisions in fog layer, there are a lot of challenges ahead, for example, stringent latency requirements, resource-constrained devices, etc. Among them, we are interested in two challenges — handling nodes update [3][4] and promising true privacy for sensitive data all over the fog layer. Due to each topic has its respective existing technologies, along with debates and disputes, the standard is not unanimous yet, in the fog computing world.

2.1. Nodes Update Handling in Fog Layer

Whenever a node to be updated, it usually satisfies one of the conditions: failed, obsolete, or compromised, no matter which, the fog layer needs to identify application exception or hardware failure and proceeds the recovery when it meets any. The major challenge is to keep the IoT system as a whole remains stable and trusted during individual node update even if it encounters certain conditions without pre-warning. The impacts of each conditions as listed in **Table 1**.

Table 1. Timing and Impacts of Fog Node Update

Update Condition	Triggered by	Impact
Failed	Hardware/ Software bug or exception	Unresponsive/ Data unavailable/ Related operations interrupted
Obsolete	System administrator (Policy or scheduled update)	Scheduled data unavailable
Compromised	Malicious node or replacement	Privacy or data lost/ Behavior changed

For the first two conditions, the affected security level is usually harmless, except influencing data accuracy or requiring re-processing by applying queue-centric treatments. What matters is the consequences of the last condition would bring: the malicious node hides somewhere, leaks the data sometime, disorders the orchestration and even jeopardizes the surrounding nodes or devices on the IoT network.

To address the above issue, a widely rolled out technology called Trusted Computing (TC) which is developed and promoted by the Trusted Computing Group. Major chip manufacturers, hardware manufacturers and operating system manufacturers forged an alliance to build in a Trusted Platform Module (TPM) with core root of trust and cryptographic modules to their products to assure the boot loader process is tamper-resistant. The scenario of TPM-integrated slightly differs for different manufacturers.

For example, “Verified boot” is used in Chrome/Android and other open source operating systems; “Secure boot” is used in Windows operating system. The technology is quite effective in protecting the system update/initialization from executing any unattested behaviors, however, it also leads to the criticism of putting too much power and control into the hands of whom designing the systems and software and finally causes anti-competitive effect, enforcing digital rights management policies and the loss of anonymity and privacy. TPM equipped computer or node is identity-unique attested and internet-availability required when processing updates, so it is possible for vendors and others who possess the ability to abuse the attestation feature to collect high sensitive data [5], no matter the user is unaware of or willing to. The issue is especially prominent for fog computing adopters — handling update on the premise of promising true privacy.

2.2. Anti-leak of Privacy Data

In a more privacy-sensitive environment the dissemination scope of sensors or devices are merely allowed to the gateways or aggregators instead of directly connecting to the internet. This kind of environment greatly reduces the possibility of collusion attacks, since all devices or sensors are pre-registered, even equipped with widely used techniques, say Iterative Filtering (IF) algorithms or reputation scoring systems [6]. When the collected data arrives the TC-enabled gateways or aggregators, as stated in section 2.1, raises new privacy leakage concerns, which may be sent to the systems or software producer imperceptible, since the internet is available. What is behind could be a machine-learning or a big data mining algorithm to perform GPS tracking and certain data fusion. Somehow, their combination, however, may uncover new meanings. Additional remote attestation could also be forced by the government’s “lawful access” proposal [7]. The unwitting leakage is possibly collected by manufactures and regulatory supervisors, not to mention the running malicious spywares or jamming attacks proactively.

Imagine a scenario that is common to industrial IoT field, thousands or more sensors and devices feedback data with filtered and processed, to dozens of corresponding center nodes. After that, the data is transmitted to a cloud data center service provider for further processing and analysis through a couple of internet-facing gateways. There are couples of problems since here: 1. The filtering and masked data were processed according to privacy rules in the local area center nodes, however, the traditional gateways just play as a network gatekeeper role to allow or block the outbound data to the pre-defined cloud destination. The masked or filtered data does not know where they will be sent to, and the gatekeeper does not know what data should be masked or filtered neither. This mismatch is usually the weak point that gives the privacy leakage an opportunity to the people with bad intentions by collecting data secretly and passing it out intermittently through the unwitting gateway before applying Privacy-Enhancing Technologies (PETs). 2. It is possible that the gateway itself sends out certain data to somewhere we don't know stealthily once detecting internet availability, especially when the gateway/server is uniquely identifiable for its remote manufacturer. 3. It is possible that the cloud data center service provider can acquire certain statistical meanings by linking the filtered data uploaded and its tenant user's source enterprise IPs or locations.

To overcome these concerns, it is of utmost importance to have an independent PETs mindset apart from IoT gateways to require rigorous oversight and security designed-in from the system outset. The PETs mechanism should be relatively independent and stands out separately from internet-facing gateways/nodes to guarantee absolutely control over any outbound data packet — even under the hypothesis that hardware gateways/nodes manufactures had embedded a certain data transmission to bypass OS level surveillance stealthily.

2.3. Zero Knowledge Proof System

A zero-kowlege proof system is to prove a statement is true without leaking extra information of statement itself. Usually there are two roles in the system, a prover and a verifier. A prover needs to convince some statement is true, for example, say x belongs to a language L is the statement we want to prove. Let P as prover, V as verifier and S as simulator. For the verifier who does not know x value if for any probabilistic polymnomial time verifier V there exists a simulator S, we can convert the zero-knowledge relationship in an equation such that

$$\forall x \in L, z \in \{0,1\}^*, View_V [\ P(x) \leftrightarrow V(x,z)\ ] = S(x,z) \tag{1}$$

where $View_V [\ P(x) \leftrightarrow V(x,z)\ ]$ is the interactions between $P(x)$ and $V(x,z)$. The existence of a simulator implies that if $x \in L$, then V cannot learn more than the fact that $x \in L$, even the x value [8].

Inspired by zero-knowledge proof, we design a scheme between the PETs processing nodes and the corresponding gateways with respect to a zero-knowledge proof system [9], which can greatly control the above threats. If we draw an analogy between PETs processing nodes and outbound fog gateways as a verifier and a prover of a zero-knowledge proof system, a malicious gateway cannot learn any sensitive statement from a source PETs processing node in a polynomial time; a malicious PETs processing node cannot convince a gateway to transmit to cloud for further processing if the privacy data has not been processed by PETs rules, such as k-Anonymity [10], l-Diversity [11] or t-Closeness [12]. The authenticating process needs the least amount of interactions between the prover and the verifier by means of the scheme that integrates the non-interactive zero knowledge (NIZK) proof [13] [14] to further decrease the validating burden. Many researches on NIZK have been proposed to improve its efficiency, for example, from the earlier work, such as the Fiat-Shamir heuristic [14] where the cryptographic hash function is introduced as a random function; recent works such as Lindell's transform of non-programmable random oracle (NPRO) model [15] that needs no random oracles to achieve efficient NIZK arguments in the common reference string

(CRS) model, and Ciampi's work [16] which combines each own's advantage of both without a random oracle. We want to defer the discussion on more efficiency improvement of NIZK and hold the controversy of which model (requiring any random oracle or not) is better, instead, put the focus on how to transform and validate PETs rules to zero-knowledge system nicely as an independent software-defined gateway (SDG) scheme which is applicable in the fog layer. Before diving into the scheme details, consider a background scenario as follow:

"De-Identification" is one typical request of PETs approaches to prevent from leaking sensitive data. One of "De-Identification" techniques is through fictitious data provided by "Data Masking". Some elaborations on the assumptions and the terms are needed here. For example, we wish to prove that the original data as shown in **Table 2** has been masked before sending out to cloud for further analysis.

Table 2. Examples of Data Masking in PETs

Algorithm	Original Data	De-Identified Data	Remark
Substitution	Stacy Martin 123-45-6789	S**** Ma**** 123-45-****	Replace the last 4 characters of each word by *
Shuffling	A123456789 13,200,423	A132547698 13,024,032	Swap position of each 2 characters-pair after the second character
Number and Data Variance	A123456789 + 11111111	A134567890	Modifies each number or value in a column to some random percentage of its real value
Generalization and Supression	Age | Name 22 | Yadale 25 | Joan 33 | Sunny 38 | Kenny	Age | Name 20 < Age 30 | * 20 < Age 30 | * 30 < Age 40 | * 30 < Age 40 | *	The combination of generalization and supression of sensitive columns

There are two types of statements can be inferred from this scenario:

- The statement of the fact: "The original data has been marked before sending out to cloud."
- The statement of the knowledge regarding to the fact: "I know the above "De-Identified Data" was indeed the masked result before sending out to cloud," even if I don't know the original data exactly.

In the column of "De-Identified Data", whether its original data being processed or not, it is easy to prove to the naked eye for the Substitution algorithm, but imperceptible for the Shuffling or the Number and Data Variance algorithm. However, this implies a significant enough difference between the two types of statements: it is possible to prove the first statement true even if we don't know the full original data. The second statement is known as a "Proof of Knowledge" which is the core logic in our scheme.

3. Proposed Scheme

Followed by **Table 2**, we pick up the algorithm “Shuffling” and its data as an example to illustrate how the proposed scheme termed as Fog Computing Zero-Knowledge Gateway (*FogComZKG*) conduct the validation process of transmission request based on zero-knowledge proof. *FogComZKG* includes three main parts of algorithms, they are named as *PETsMasker*, *DIGESTER* in prover’s side and *ZKVerifier* in verifier’s side, respectively. Begin with the assumption that the rule of PETs needs to be transformed into a very specific format and to be signed by one of the keys in the key pair of prime number and a generator of a cyclic group [17] of prime-order format in Claus-Peter Schnorr’s signature theory [18].

A high-level procedure is presented in **Fig. 2**, inspired by zero-knowledge proof, let PETs processing node be a prover for its sending data content and let *ZKVerifier* be a verifier agent for outbound request approver of internet gateway.

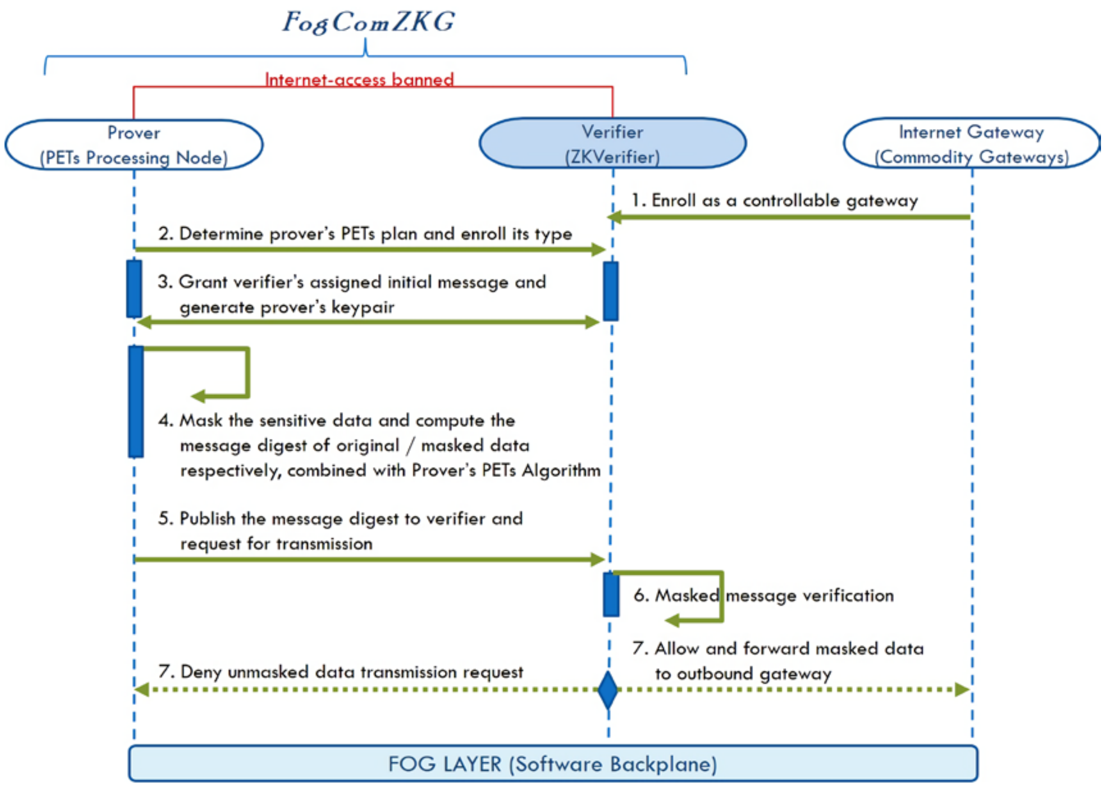
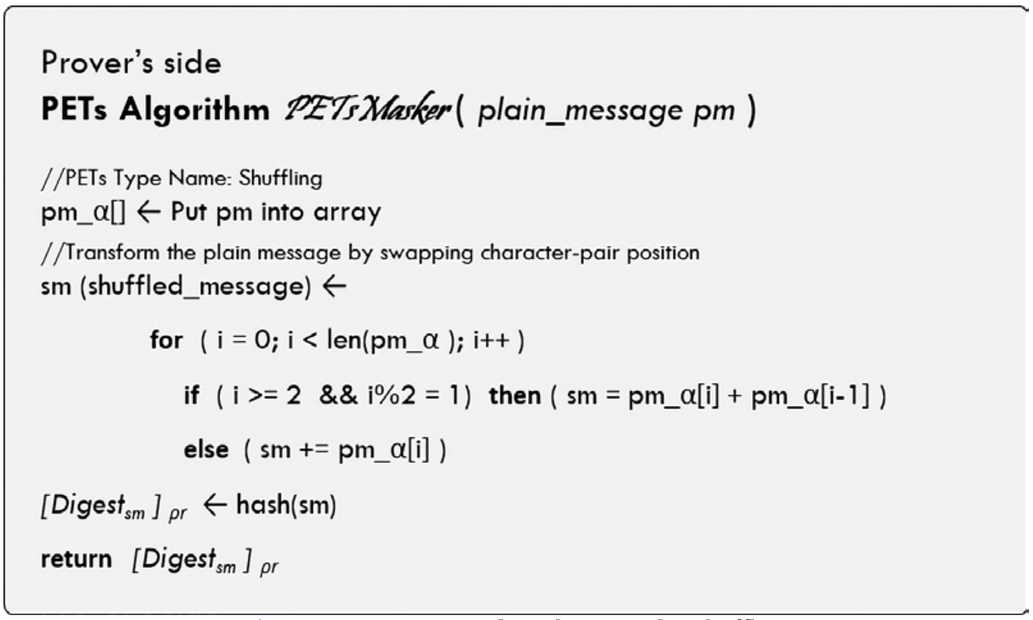

FogComZKG
Internet-access banned
Prover (PETs Processing Node)
Verifier (ZKVerifier)
Internet Gateway (Commodity Gateways)
1. Enroll as a controllable gateway
2. Determine prover’s PETs plan and enroll its type
3. Grant verifier’s assigned initial message and generate prover’s keypair
4. Mask the sensitive data and compute the message digest of original / masked data respectively, combined with Prover’s PETs Algorithm
5. Publish the message digest to verifier and request for transmission
6. Masked message verification
7. Allow and forward masked data to outbound gateway
7. Deny unmasked data transmission request
FOG LAYER (Software Backplane)

Fig. 2. High level sequence flow chart of FogComZKG

Step 1: The existing commodity gateways are based on hardware device or software-defined module. The enrollment is required to have proxy permission to generate effective operating configurations or commands (allow/deny outbound network traffic on specific interfaces for specific request) to internet-facing gateway.

Step 2: PETs processing node as a prover must be enrolled to a verifier and let the verifier know what the applied PETs plan is (Only disclose the masker type later in Step 4 instead of the whole masker details.) for each data message when it requests for transmission. We called it *PETsMasker* as shown in **Fig. 3** which is a collection of any desired PETs algorithms, shuffling here is just one of algorithm examples.

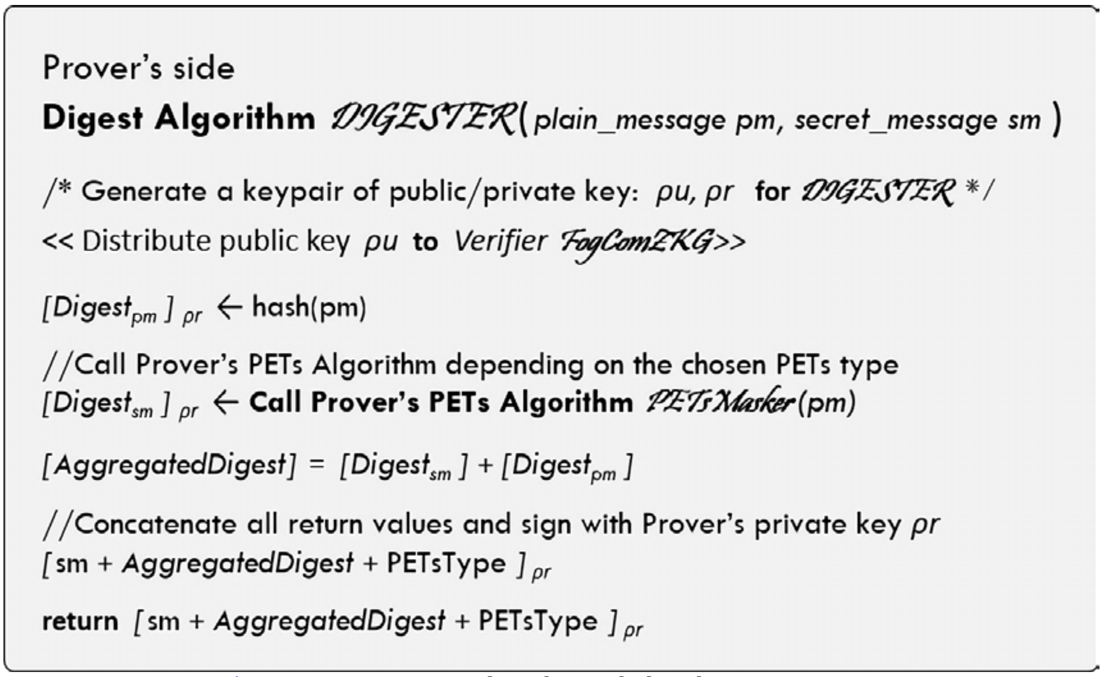


```
Prover's side
PETs Algorithm PETsMasker( plain_message pm )

//PETs Type Name: Shuffling
pm_α[] ← Put pm into array
//Transform the plain message by swapping character-pair position
sm (shuffled_message) ←
        for ( i = 0; i < len(pm_α ); i++ )
            if ( i >= 2 && i%2 = 1) then ( sm = pm_α[i] + pm_α[i-1] )
            else ( sm += pm_α[i] )
$[Digest_{sm}]_{pr}$ ← hash(sm)
return $[Digest_{sm}]_{pr}$
```

Fig. 3. Prover's PETs Algorithm Sample: Shuffling

Step 3: *ZKVerifier* as a verifier who needs to choose some value as its challenge based on its random oracle mechanism (as opposed to the value from the prover) and then outputs a specific initial message for Step 2. Prover generates his owned keypair for later signature.

Step 4: Some random value corelated to the sensitive data is included as an initiative of keypair. Prover hashes the original data and the masked one respectively and signs the combination of each as a message digest with prover's private key. The signed digest is not merely a proof of the knowledge of the sensitive data but also a signature by someone who really owns the sensitive data. A novel trick of *DIGESTER* differs from the known digital signature: the digest is an aggregation that look like a single digest referring to the original data and the masked one, respectively. The prover also needs to specify which part of the aggregated message in the digest is for the masked data explicitly. *DIGESTER* combines the specified *PETsMasker* algorithm as shown in **Fig. 4** and is ready as the publication for next step.



```
Prover's side
Digest Algorithm DIGESTER( plain_message pm, secret_message sm )

/* Generate a keypair of public/private key: pu, pr for DIGESTER */
<< Distribute public key pu to Verifier FogComZKG>>

$[Digest_{pm}]_{pr}$ ← hash(pm)

//Call Prover's PETs Algorithm depending on the chosen PETs type
$[Digest_{sm}]_{pr}$ ← Call Prover's PETs Algorithm PETsMasker(pm)

[AggregatedDigest] = $[Digest_{sm}]$ + $[Digest_{pm}]$

//Concatenate all return values and sign with Prover's private key pr
$[sm + AggregatedDigest + PETsType]_{pr}$

return $[sm + AggregatedDigest + PETsType]_{pr}$
```

Fig. 4. Prover's Digest Algorithm with the Chosen PETs Type

Step 5: Prover publishes the masked data along with the aggregated message digest in a bundle to verifier to ask for outbound permission when it needs to transmit the data to internet cloud data center. However, a prover has more than one possibility for the provided publication.

Publication 1:

Nonsense data + Message digest
or
Any data without Message digest

Publication 2:

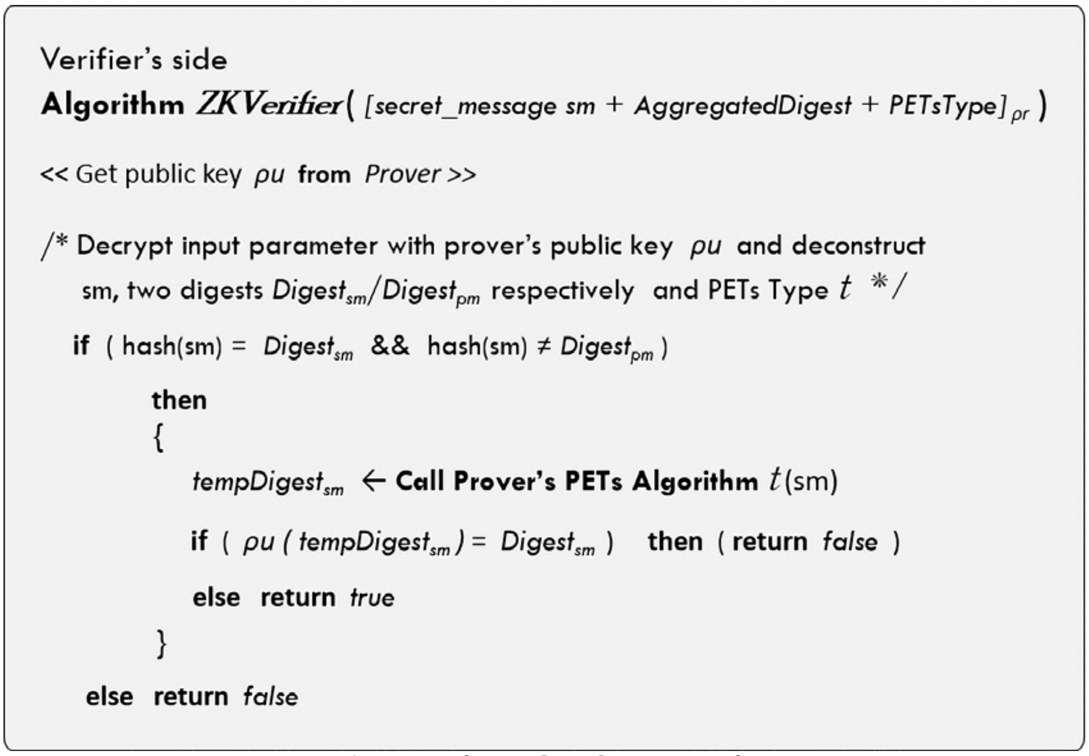
Original data + Message digest

Publication 3:

Masked data + Message digest

Step 6: Verifier's algorithm is the crucial gatekeeper before reaching out to the real internet-facing gateway. As presented in **Fig. 5**, it decrypts the ciphered input with prover's public key and deconstructs it as declared data, digital signature and PETs type. Verifier then extracts the masked part in the digital signature and hashes the declared data *sm* again, if they are identical which proves the data integrity and its valid source (rules out the possibility of Publication 1). The first-round traditional verification implies the capability in avoiding from data leakage of any unauthorized and malicious transmission request. It also applicable in solving fog nodes update challenge of whether the declared update executable is from its authentic manufacturer once the updating request convinced the verifier herein before performing installation.

Owing to there are two more unsure publications that a prover can provide, we need further verifications to confirm either the declared data is the one masked (as stated in Publication 3) or the one unmasked (as stated in Publication 2). The challenge is the original sensitive data *pm* should not only be zero-knowledge to verifier but also prover-owned provable. *FogComZKG* calls back the same prover's PETs algorithm with inputting the declared secret message in a novel way and decrypts the returning value, if it happens to be the digest of the secret message, that implies the declared secret is not really as secret as it claimed, because the digest of the re-masked real secret should not be the same as the digest of the masked sensitive data; otherwise, the declared secret message is indeed a secret. Prover can only convince the verifier only when he is an honest prover, in the meantime, verifier stays zero-knowledge of the secret data during the entire process.



```
Verifier's side
Algorithm ZKVerifier( [secret_message sm + AggregatedDigest + PETsType]$_{pr}$ )

<< Get public key $\rho u$ from Prover >>

/* Decrypt input parameter with prover's public key $\rho u$ and deconstruct
   sm, two digests $Digest_{sm}$/$Digest_{pm}$ respectively and PETs Type $t$ */
  if ( hash(sm) = $Digest_{sm}$ && hash(sm) $\neq$ $Digest_{pm}$ )
      then
      {
          $tempDigest_{sm}$ ← Call Prover's PETs Algorithm $t$(sm)
          if ( $\rho u$ ( $tempDigest_{sm}$ ) = $Digest_{sm}$ )  then ( return false )
          else return true
      }
   else return false
```

Fig. 5. Verifier's Algorithm: ZKVerifier

Step 7: Verifier acts regarding to the verified result: *FogComZKG* filters and composes corresponding commands according to the request, then forwards the masked data traffic to internet gateway once being convinced, or just denies the request.

Command example: The only packets allowed to the interface must be from source IP of FogNode01 (172.16.2.10). The extended access list named FogNode01-batch01 filters incoming packets. The access list permits http packets from the source 172.16.2.10 to a certain public cloud datacenter network 40.79.160.0.0 and denies all other TCP packets. The access list denies all other IP packets and performs logging of packets passed or denied by that entry.

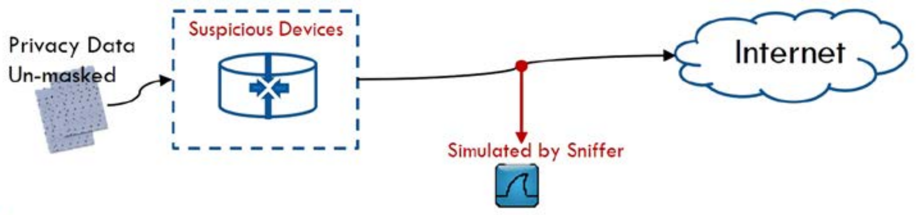
```
ip access-list standard Internet-filter
permit 172.16.2.10
ip access-list extended FogNode01-batch01
```

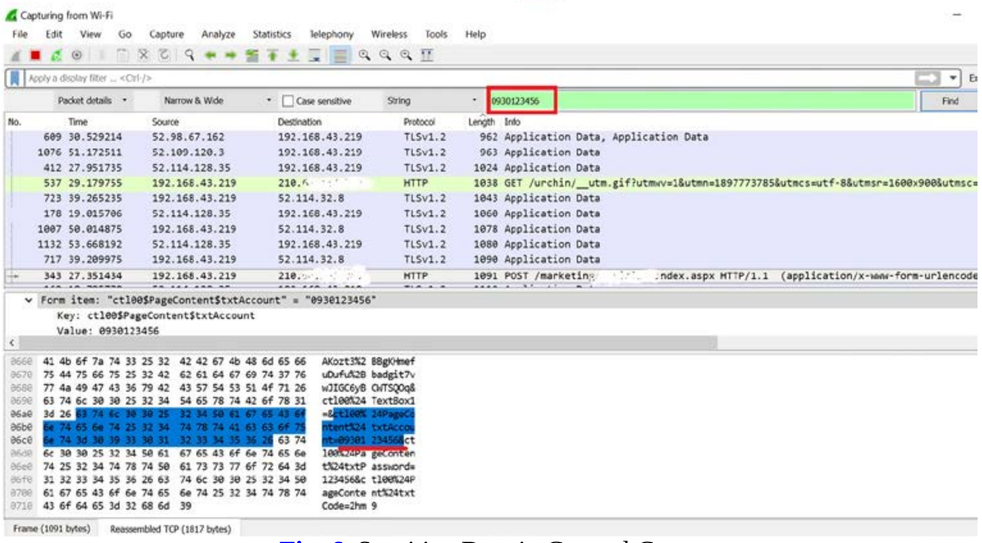
permit tcp 172.16.2.10 40.79.160.0.0 0.0.255.255 eq http
deny tcp any any
deny ip any any

4. Evaluation of FogComZKG Scheme

We implemented the main algorithms of prover's and verifier's steps in section 3 to examine its effect. The work is designed for the domian such as industrial IoT manufacturers or financial services who on the one hand collect information centrally from distributed devices or registered clients but require the second time information transmission or exchange to cloud service provider for further analysis. The security threats may happen on hardware network device itself stealthily, attack by sniffer or by mistake. In this section, we design a case study to simulate the sniffing attack against the information leak during the re-distribution process. The case simulates a TPM-based L3 switch eavesdropping certain privacy data without properly masked or encrypted and forwarding the data passing through it to its manufactor (the manufactor's IP is usually legal in the allowed list for checking updates purpose) along with the allowed service destination (legal in the access control list), we call it "*privacy smuggling*". The forwarding transmission is hard to be detected owing to the fact of incidental transimission mixing up with normal traffic. We then launch a sniffing attack to spy upon privacy data sent by a fog node netwrok and present how our work effectively protect the privacy with shuffling algorithm in **Fig. 6** and **Fig. 7**.

For instance, the sensitive data is 0930123456, if not being shuffled, it can be sniffed just as its original content in general case.

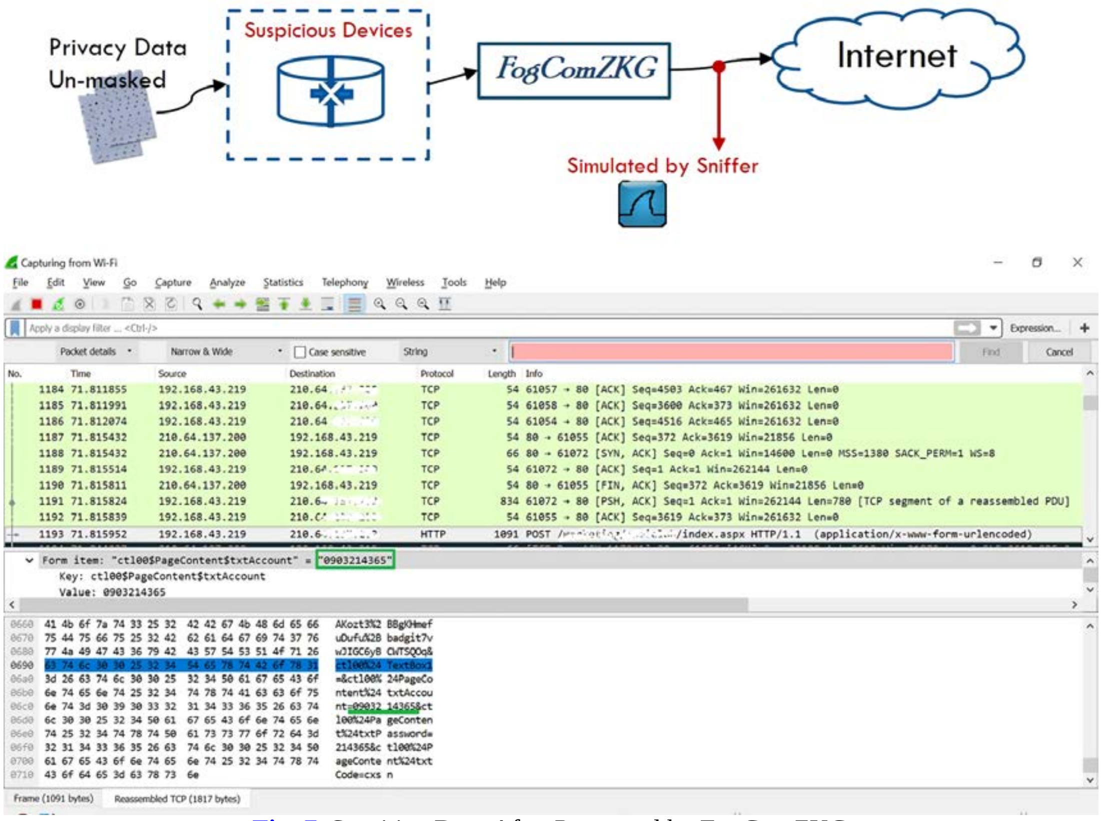



Capturing from Wi-Fi
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Packet details
Narrow & Wide
Case sensitive
String
0930123456
Find

```
No.   Time       Source          Destination     Protocol Length Info
 609 30.529214  52.98.67.162    192.168.43.219  TLSv1.2   962 Application Data, Application Data
1076 51.172511  52.109.120.3    192.168.43.219  TLSv1.2   963 Application Data
 412 27.951735  52.114.128.35   192.168.43.219  TLSv1.2  1024 Application Data
 537 29.179755  192.168.43.219  210.6[redacted] HTTP     1038 GET /urchin/__utm.gif?utmwv=1&utmn=1897773785&utmcs=utf-8&utmsr=1600x900&utmsc=
 723 39.265235  192.168.43.219  52.114.32.8     TLSv1.2  1043 Application Data
 178 19.015706  52.114.128.35   192.168.43.219  TLSv1.2  1060 Application Data
1007 50.014875  192.168.43.219  52.114.32.8     TLSv1.2  1078 Application Data
1132 53.668192  52.114.128.35   192.168.43.219  TLSv1.2  1080 Application Data
 717 39.209975  192.168.43.219  52.114.32.8     TLSv1.2  1090 Application Data
 343 27.351434  192.168.43.219  210.[redacted]  HTTP     1091 POST /marketin[redacted]ndex.aspx HTTP/1.1  (application/x-www-form-urlencode

  Form item: "ctl00$PageContent$txtAccount" = "0930123456"
      Key: ctl00$PageContent$txtAccount
      Value: 0930123456

0660  41 4b 6f 7a 74 33 25 32  42 42 67 4b 48 6d 65 66   AKozt3%2 BBgKHmef
0670  75 44 75 66 75 25 32 42  62 61 64 67 69 74 37 76   uDufu%2B badgit7v
0680  77 4a 49 47 43 36 79 42  43 57 54 53 51 4f 71 26   wJIGC6yB CWTSQOq&
0690  63 74 6c 30 30 25 32 34  54 65 78 74 42 6f 78 31   ctl00%24 TextBox1
06a0  3d 26 63 74 6c 30 30 25  32 34 50 61 67 65 43 6f   =&ctl00% 24PageCo
06b0  6e 74 65 6e 74 25 32 34  74 78 74 41 63 63 6f 75   ntent%24 txtAccou
06c0  6e 74 3d 30 39 33 30 31  32 33 34 35 36 26 63 74   nt=09301 23456&ct
06d0  6c 30 30 25 32 34 50 61  67 65 43 6f 6e 74 65 6e   l00%24Pa geConten
06e0  74 25 32 34 74 78 74 50  61 73 73 77 6f 72 64 3d   t%24txtP assword=
06f0  31 32 33 34 35 36 26 63  74 6c 30 30 25 32 34 50   123456&c tl00%24P
0700  61 67 65 43 6f 6e 74 65  6e 74 25 32 34 74 78 74   ageConte nt%24txt
0710  43 6f 64 65 3d 32 68 6d  39                        Code=2hm 9

Frame (1091 bytes)   Reassembled TCP (1817 bytes)
```

Fig. 6. Sensitive Data in General Case

The original sensitive data is 0930123456 and being shuffled as 0903214365 forcibly by FogComZKG pre-registered rule as long as it needs to transfer to internet. Although it can be sniffed, the content is shuffled. Pre-registered rule can be changed depends on requirement.



```
No.   Time       Source          Destination      Protocol  Length Info
1184 71.811855  192.168.43.219  210.64[redacted]    TCP        54 61057 → 80 [ACK] Seq=4503 Ack=467 Win=261632 Len=0
1185 71.811991  192.168.43.219  210.64.[redacted]   TCP        54 61058 → 80 [ACK] Seq=3600 Ack=373 Win=261632 Len=0
1186 71.812074  192.168.43.219  210.64[redacted]    TCP        54 61054 → 80 [ACK] Seq=4516 Ack=465 Win=261632 Len=0
1187 71.815432  210.64.137.200  192.168.43.219   TCP        54 80 → 61055 [ACK] Seq=372 Ack=3619 Win=21856 Len=0
1188 71.815432  210.64.137.200  192.168.43.219   TCP        66 80 → 61072 [SYN, ACK] Seq=0 Ack=1 Win=14600 Len=0 MSS=1380 SACK_PERM=1 WS=8
1189 71.815514  192.168.43.219  210.64[redacted]    TCP        54 61072 → 80 [ACK] Seq=1 Ack=1 Win=262144 Len=0
1190 71.815811  210.64.137.200  192.168.43.219   TCP        54 80 → 61055 [FIN, ACK] Seq=372 Ack=3619 Win=21856 Len=0
1191 71.815824  192.168.43.219  210.6[redacted]     TCP       834 61072 → 80 [PSH, ACK] Seq=1 Ack=1 Win=262144 Len=780 [TCP segment of a reassembled PDU]
1192 71.815839  192.168.43.219  210.[redacted]      TCP        54 61055 → 80 [ACK] Seq=3619 Ack=373 Win=261632 Len=0
1193 71.815952  192.168.43.219  210.6[redacted]     HTTP     1091 POST /[redacted]/index.aspx HTTP/1.1  (application/x-www-form-urlencoded)

Form item: "ctl00$PageContent$txtAccount" = "0903214365"
    Key: ctl00$PageContent$txtAccount
    Value: 0903214365

0660  41 4b 6f 7a 74 33 25 32  42 42 67 4b 48 6d 65 66   AKozt3%2 BBgKHmef
0670  75 44 75 66 75 25 32 42  62 61 64 67 69 74 37 76   uDufu%2B badgit7v
0680  77 4a 49 47 43 36 79 42  43 57 54 53 51 4f 71 26   wJIGC6yB CWTSQOq&
0690  63 74 6c 30 30 25 32 34  54 65 78 74 42 6f 78 31   ctl00%24 TextBox1
06a0  3d 26 63 74 6c 30 30 25  32 34 50 61 67 65 43 6f   =&ctl00% 24PageCo
06b0  6e 74 65 6e 74 25 32 34  74 78 74 41 63 63 6f 75   ntent%24 txtAccou
06c0  6e 74 3d 30 39 30 33 32  31 34 33 36 35 26 63 74   nt=09032 14365&ct
06d0  6c 30 30 25 32 34 50 61  67 65 43 6f 6e 74 65 6e   l00%24Pa geConten
06e0  74 25 32 34 74 78 74 50  61 73 73 77 6f 72 64 3d   t%24txtP assword=
06f0  32 31 34 33 36 35 26 63  74 6c 30 30 25 32 34 50   214365&c tl00%24P
0700  61 67 65 43 6f 6e 74 65  6e 74 25 32 34 74 78 74   ageConte nt%24txt
0710  43 6f 64 65 3d 63 78 73  6e                        Code=cxs n
```

Fig. 7. Sensitive Data After Processed by FogComZKG

In addition to security, efficiency is one essential factor to determine feasibility. We also examine the performance when using different hash functions in producing digital signatures regarding to different message sizes. In the experiment result as shown in **Table 3**, the process time of the shuffling operation was excluded, owning to the processing time of various PETs rules differs and users always have freedoms to apply different PETs rule within our scheme. The examination ran on a server with a 2.6 GHz Intel® i7 6600U 4 processors, 16 GB RAM based on Microsoft Windows 10 Enterprise Build 16299. The result is not to our surprise, MD5 is the fastest, the SHA families are approximate in order of their complexity. In our scheme, during the interaction it requires at least three times hashes, there are two times of hashes from prover and one time of hash from verifier. It is a limit so far as an interactive zero-knowledge protocol in our scheme.

Table 3. FogComZKG Performance Regarding to Different Hash Types and Message Sizes

Process Time (ms) of FogComZKG Prover& Verifier Operation (Excluded the PETs rule part process time)	512 KB	1 MB	10 MB	256 MB	512 MB	1024 MB	
Used Hash Functions							Hashed Sizes
MD5	33	33	34	46	51	137	32 chars
SHA-1	40	41	42	52	54	145	40 chars
SHA-256	58	60	63	76	233	912	64 chars
SHA-512	59	62	65	88	604	2213	128 chars

5. Conclusion

In this paper, we propose a software-defined gateway scheme to help define a fog layer component that we consider a novel candidate of the evolving fog computing standards, especially for the scenarios of quite sensitive message protection and internet-controlled, for example, industrial IoT or banking. The solution provides a capability of partial masking, pluses the privacy enhancement technologies in its design time, instead of encrypting sensitive data altogether recklessly since it obstructs further cloud application, the scheme enforces the protection policy firmly applied on message before leaving fog layer for cloud. The scheme also introduces a novel approach to stay neutral and zero-knowledge to the sensitive data and workable without internet connectivity. It greatly reduces the doubt of data leakage of itself and removes the potential leakage possibility by other devices, such as TPM-enabled servers, firewall or gateways.

6. Future Work

In future work, we will extend our zero-knowledge protocol from current interactive way to non-interactive way to avoid increasing hashing overhead as the number of the messages increases. Artificial intelligence technologies will be planned and leveraged to extend the protecting coverage of the scheme. We may also validate some part of the scheme, such as hash functions, with Field Programmable Gate Array (FPGA) implementations [19][20][21] or blocked RAM to further enhance its efficiency.

References

[1] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu, "Edge Computing: Vision and Challenges," *IEEE INTERNET OF THINGS JOURNAL*, vol. 3, no. 5, pp-637-646, Oct. 2016. Article (CrossRef Link)

[2] Stefan Nastic, Hong-Linh Truong, and Schahram Dustdar, "SDG-Pro: a programming framework for software-defined IoT cloud gateways," *Journal of Internet Services and Applications*, 6:21, Oct. 2015. Article (CrossRef Link)

[3] Eren Balevi, and Richard D. Gitlin, "Optimizing the Number of Fog Nodes for Cloud-Fog-Thing Networks," *Networking and Internet Architecture*, 4 Jan. 2018. Article (CrossRef Link)

[4] M. Weißbach, N. Taing, M. Wutzler, T. Springer, A. Schill and S. Clarke, "Decentralized coordination of dynamic software updates in the Internet of Things," in *Proc. of 2016 IEEE 3rd World Forum on Internet of Things (WF-IoT), Reston*, VA, pp. 171-176, 2016. Article (CrossRef Link)

[5] Trusted Computing Article (CrossRef Link)

[6] E. Choudhari, K. D.Bodhe, S. M. Mundada, "Secure data aggregation in WSN using iterative filtering algorithm," in *Proc. of 2017 International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, Bangalore, pp. 1-5, 2017. Article (CrossRef Link)

[7] G. Hariton, H. Palihapitya, "Should Consumers Trust Trusted Computing?," *2005 Public Interest Advocacy Centre (PIAC).*

[8] Goldwasser, S.; Micali, S.; Rackoff, C., "The knowledge complexity of interactive proof systems," (PDF), *SIAM Journal on Computing, Philadelphia: Society for Industrial and Applied Mathematics*, 18 (1): 186-20, 1989. Article (CrossRef Link)

[9] Shafi Goldwasser, Silvio Micali, and Charles Rackoff, "The Knowledge Complexity of Interactive Proof-Systems," in *Proc. of Proceedings of the seventeenth annual ACM symposium on Theory of computing (STOC 1985)*, pp. 291-304, 1985. Article (CrossRef Link)

[10] L. Sweeney. "k-Anonymity: a model for protecting privacy," *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10 (5), 557-570. Paper: 14, 2002. Article (CrossRef Link)

[11] A. Machanavajjhala, D. Kifer, J. Gehrke, M. Venkitasubramaniam, "l-Diversity: Privacy Beyond k-Anonymity," in *Proc. of 22nd International Conference on Data Engineering (ICDE'06)*, Atlanta, GA, USA, pp. 24-24, 2006. Article (CrossRef Link)

[12] N. Li, T. Li and S. Venkatasubramanian, "t-Closeness: Privacy Beyond k-Anonymity and l-Diversity," in *Proc. of 2007 IEEE 23rd International Conference on Data Engineering*, Istanbul, pp. 106-115, 2007. Article (CrossRef Link)

[13] Manuel Blum, Paul Feldman, and Silvio Micali, "Non-Interactive Zero-Knowledge and Its Applications," in *Proc. of Proceedings of the twentieth annual ACM symposium on Theory of computing (STOC 1988)*, pp. 103-112, 1988. Article (CrossRef Link)

[14] Amos Fiat and Adi Shamir, "How to Prove Yourself: Practical Solutions to Identification and Signature Problems," in *Proc. of CRYPTO 1986*, pp. 186-194, 1986. Article (CrossRef Link)

[15] Yehuda Lindell, "An Efficient Transform from Sigma Protocols to NIZK with a CRS and Non-Programmable Random Oracle," *TCC 2015: Theory of Cryptography*, pp. 93-109, 2015. Article (CrossRef Link)

[16] Ciampi M., Persiano G., Siniscalchi L., Visconti I, "A Transform for NIZK Almost as Efficient and General as the Fiat-Shamir Transform Without Programmable Random Oracles," *Theory of Cryptography. TCC 2016. Lecture Notes in Computer Science*, vol 9563. Springer, Berlin, Heidelberg, 2016. Article (CrossRef Link)

[17] Hazewinkel, Michiel, ed., "Cyclic group," *Encyclopedia of Mathematics, [1994] Springer Science+Business Media B.V. / Kluwer Academic Publishers*, ISBN 978-1-55608-010-4, 2001. Article (CrossRef Link)

[18] C.P. Schnorr, "Efficient identification and signatures for smart cards," in *Proc. of G. Brassard, ed. Advances in Cryptology—Crypto '89, 239-252, Springer-Verlag*. Lecture Notes in Computer Science, nr 435, 1990. Article (CrossRef Link)

[19] F. Kahri, H. Mestiri, B. Bouallegue and M. Machhout, "Efficient FPGA hardware implementation of secure hash function SHA-256/Blake-256," in *Proc. of 2015 IEEE 12th International Multi-Conference on Systems, Signals & Devices (SSD15)*, Mahdia, pp. 1-5, 2015. Article (CrossRef Link)

[20] Shi Z., Ma C., Cote J., Wang B., "Hardware Implementation of Hash Functions. In: Tehranipoor M., Wang C. (eds) Introduction to Hardware Security and Trust," *Springer*, New York, NY, 2012. Article (CrossRef Link)

[21] Latif K., Tariq M., Aziz A., Mahboob A. *(2012) Efficient Hardware Implementation of Secure Hash Algorithm (SHA-3) Finalist - Skein. In: Sambath S., Zhu E. (eds) Frontiers in Computer Education. Advances in Intelligent and Soft Computing*, vol 133. Springer, Berlin, Heidelberg, 2012. Article (CrossRef Link)

Te-Yuan Lin is a PhD candidate in Computer Science and Information Engineering of National Taiwan University, Taipei, Taiwan, R.O.C. Before that, he earned his master's degree in computer information system from Baruch College, the City University of New York. His academic research explores big data and databases efficiency, cloud computing, cryptography and security. He regularly speaks on technical forums and conferences of best practices about cloud computing and security. He served as a technical consultant for Microsoft for several years. In his recent research, he pays more focus on the secure interactions of Internet of Things, fog computing and cloud computing applications.

Chiou-Shann Fuh received the MS degree in computer science from the Pennsylvania State University, University Park, PA, in 1987, and the PhD degree in computer science from Harvard University, Cambridge, MA, in 1992. He was with AT&T Bell Laboratories and engaged in performance monitoring of switching networks from 1992 to 1993. He is a full professor in Department of Computer Science and Information Engineering, National Taiwan University, Taipei. His current research interests include digital image processing, computer vision, pattern recognition, mathematical morphology, and their applications to defect inspection, industrial automation, digital video camcorder, surveillance camera, and camera module such as color interpolation, auto exposure, auto focus, and color management.