Chronos: Towards Securing System Time in the Cloud for Reliable Forensics Investigation

Shams Zawoad and Ragib Hasan
{zawoad, ragib}@cis.uab.edu
Department of Computer and Information Sciences
University of Alabama at Birmingham, AL 35294, USA

Abstract—**In digital forensics investigations, the system time of computing resources can provide critical information to implicate or exonerate a suspect. In clouds, alteration of the system time of a virtual machine (VM) or a cloud host machine can provide unreliable time information, which in turn can mislead an investigation in the wrong direction. In this paper, we propose *Chronos*[1] to secure the system time of cloud hosts and VMs in an untrusted cloud environment. Since it is not possible to prevent a malicious user or a dishonest insider of a cloud provider from altering the system time of a VM or a host machine, we propose a tamper-evident scheme to detect this malicious behavior at the time of investigation.**

We integrate Chronos with an open-source cloud platform – OpenStack and evaluate the feasibility of Chronos while running 20 VMs on a single host machine. Our test results suggest that Chronos can be easily deployed in the existing cloud with very low overheads, while achieving a high degree of trustworthiness of the system time of the cloud hosts and VMs.

I. Introduction

Throughout history, it has been observed that general technological developments have continually created new opportunities for criminal activities. This is also true for the emergence of cloud computing. While the high degree of scalability, very convenient pay-as-you-go service, and low cost computing provided by clouds drive the rapid adoption of clouds [1], [2], [3], [4], [5], these features can motivate a malicious individual to launch attacks from machines inside a cloud [6], [7], or use the cloud to store contraband documents [8], [9]. In the investigation process of such cloud-based attacks, the time associated with digital evidence can be very crucial to discharge or convict a suspect [10].

It was reported that the primary suspect of a 1995 homicide case claimed that he was at work at the time of the murder and his alibi was the last configuration time of a Fastpath network device [11]. Since the suspect had full control over the management console of the device, investigators believed that the suspect reset the time on the device from the management console that supported his alibi. The management console was not collected during the initial search and seizure and hence, there was a high degree of uncertainty in the timestamp. Eventually, the suspect's alibi could not be confirmed and he was convicted of the murder.

A similar situation can occur with the cloud, where a crime is committed directly using the cloud, or where cloud activities can be used as evidence for any other crime. Therefore, we need to make sure that the system clocks of cloud host machines or virtual machines have not been tampered with. However, an attacker can change a VM's system clock before launching an attack and later reset it to the original time. The following hypothetical scenario illustrates the specific problem that we intend to solve:

Bob runs a successful online business website. Mallory, a competitor to Bob's business, launched a distributed denial of service (DDoS) attack using VMs rented from a cloud provider, CloudCo. The DDos attack made Bob's website unavailable on March 1st, 2015 from 10:00 am to 11:00 am. By analyzing Bob's website record, Charlie, a forensic investigator found that the website was flooded by some IP addresses that were owned by CloudCo. Eventually, Charlie issued a subpoena to CloudCo to acquire the activity logs of the VMs that owned those IPs. However, Mallory was maintaining a backdated system clock in her VMs at the time of attack, so Charlie found that on March 1st, 2015 from 10:00 am to 11:00, there was no communication between Mallory's VMs and Bob's website. With this contradictory data, Charlie further requested for host machine's logs. However, before launching the DDoS attack, Mallory was able to collude with a CloudCo employee and that employee set the same backdated time in the host machine. Under this circumstance, Charlie would not be able to prove Mallory as the attacker.

To mitigate the challenges discussed in the hypothetical attack scenario, we propose *Chronos*, which ensures the trustworthiness of the system time of malicious cloud hosts and VMs using a tamper-evident cryptographic scheme. To ensure the desired security properties, we introduce a dedicated Chronos Server (CS) with the existing OpenStack [12] architecture. First, we run a secure pairwise time verification phase among the cloud host, VM, and CS, where each entity verifies others' system time. Second, information about the verification phase is stored securely within CS using cryptographic hash-chain of the verification results. After some certain epoch, the proof of this hash-chain is published on the Internet. The hash-chain and the proof of a chain ensure the detection of system time alterations when a VM, host, and CS all are compromised. We evaluate the feasibility of Chronos and identify various system properties on a private cloud built on top of OpenStack.

Our Contributions:

• We propose Chronos to secure the system time of cloud hosts and VMs in an untrusted cloud environment. The existing works [13], [14] depend on the "happened before" relation [15] and consider the cloud service provider (CSP) as honest. However, not all the events occurred inside the cloud have the

[1]Chronos: comes from ancient Greek, referred as the God of time.

happened before relation between them. Additionally, the cloud is considered as untrusted in contemporary research works [16], [17], [18], [19], [20], [21]. Chronos does not depend on the "happened before" relation and also ensures the trustworthiness of system time even when a CSP gets compromised or becomes dishonest. Hence, Chronos increases the reliability of digital forensics investigation in clouds.

• We rigorously analyze the threats on maintaining trustworthy system time in clouds and present a threat model, which will facilitate future research in this area.

• We evaluate the feasibility of Chronos by integrating it with a private cloud built on top of OpenStack. Our test results suggest that when a host machine runs in its full capacity, executing the timestamp verification cycle in every 60 seconds yields less than 1% system overhead for the cloud host machine. Our test results also show that our implementation of Chronos offers high degree of stability and fault tolerance.

Organization: Section II presents the related research works. Section III describes the system model for a secure and trustworthy system time. Section IV presents our proposed scheme. Section V provides the security analysis of Chronos. Section VI presents the experimental results. Section VII discusses several critical issues about securing system time and finally, we conclude in Section VIII.

II. Related Work

Secure time synchronization and ordering of events in a distributed system have been discussed by researchers from different perspectives [10], [14], [15], [22], [23], [24]. To identify out-of-sequence events in a distributed multi-process system, Lamport *et al.* first introduced the *happened before* relation between events [15] and proposed an algorithm to synchronize events occurring in a distributed system. Later, Gladyshev *et al.* defined *event time bounding* for digital forensics using the *happened before* relation [13] and proposed an algorithm to calculate time intervals of events by considering their causal connections with other events whose time is known. To determine the temporal behavior of a suspect computer, Schatz *et al.* correlates timestamped events found on the suspect computer with timestamped events from a more reliable, corroborating source [25]. Stephens proposed a model to relate timestamps taken from multiple timelines [26]. In this model, a base clock is set to UTC, and subordinate clocks are defined in terms of skews from parent clocks with additional skews further generated from time drift rates. There has been substantial research on secure timestamp synchronization in the field of wireless sensor networks [27], [22], [28], which targeted to secure communications and protect timing delay attacks by an external intruder.

Secure timestamps were discussed in the literature of proving the existence of a digital document prior to a specific point in time [23], [29], [30], [31], [32]. In [32], Harber *et al.* proposed that whenever a user needs a document to be time-stamped, he/she sends the hash of the document to a time-stamping service (TSS). The TSS then appends the date and time with the hash, signs the compound object, and returns to the user. To defend forward-dated or back-dated timestamp, TSS includes bits from previous sequence of client request in the timestamp certificate. Later, Bonnecaze *et al.* proposed a document time-stamping scheme based on distributed TSS rather than a single TSS, which ensures trustworthy time-stamping when less than 1/3 of the total TSS are malicious [30].

Thorpe *et al.* developed a log auditor by using the *happened before* relation in clouds to detect temporal inconsistencies in a VM's timeline [14]. According to their scheme, if a VM event A has a *happened before* relation with a VM event B, while the VM kernel log suggests that timestamp T_B of B precedes timestamp T_A of A, then T_A and T_B are inconsistent. However, not all the events occurred inside clouds have the happened before relation between them.

The proposed systems also do not consider the insider threats when an employee of a CSP, or the CSP as a whole is dishonest. Hence, the existing systems cannot solve the problem that we address in this paper.

III. System Model

A. Definition of terms

• *Timestamp:* Timestamp refers to the time given by the system clock. The trustworthiness of the time attached with any type of cloud evidence depends on the security of the timestamp service.

• *Chronos Server (CS):* The Chronos Server (CS) verifies the timestamp of cloud hosts and VMs. The CS is under the control of the CSP and is responsible for storing all information about the timestamp verification phase.

• *Attestation (A):* An attestation of a timestamp is a verifiable statement from an entity supporting the truth value of the timestamp of another entity.

• *Certification (C):* A certification is a verifiable statement from one entity supporting the correctness of an attestation provided by another entity. An attestation statement is preserved within a certification statement.

• *Certification-Chain (CC):* The certification-chain maintains the integrity of certifications using the hash-chain scheme.

• *Certified-Timestamp (CT):* The Chronos server preserves the information of attestations, certifications, and certification-chains in the form of a certified-timestamp.

• *Proof of Time (POT):* The POT is a cryptographic proof, which is published to the Internet and is used to verify the integrity of certified-timestamps.

• *Requestor:* A Requestor refers to the entity that wants to verify its own timestamp.

• *Verifier:* A verifier checks the timestamp of a requestor based on its own timestamp and provides an attestation to the requestor.

• *Certifier:* A certifier validates an attestation statement issued by a verifier and provides a certification about the validity of the attestation.

• *Auditor:* An auditor is the court authority, which verifies the correctness of the timestamps for a particular time range.

• *Intruder:* An intruder can be any malicious person including an insider from the CSP, who wants to attack the attestation and certification procedures.

B. Attacker's Capability

While designing Chronos, we consider the following assumptions regarding the capabilities of adversaries:

1. In our threat model, a malicious user is capable of changing the system time of the virtual machines that he/she rents from a cloud service provider. An adversary, who is in control of a compromised host or CS, can change the system time of these machines. An external attacker or a dishonest employee of the cloud provider, who colludes with a malicious user or a dishonest investigator, can compromise a host and a CS.

2. The host and the CS can be honest at the time of timestamp verification, but can collude later with users or investigators to remove, reorder, or deny attestations and certifications. An investigator does not participate in the timestamp verification phase but can alter the attestation and certification information after acquiring these from the cloud.

3. Users and CSPs have access to the network and hence are capable of introducing an intentional delay in the timestamp verification process.

4. An adversary cannot spoof the system time used in the timestamp verification process. To do so, it requires the man in the middle (MITM) attack while invoking a system call to get the current timestamp. We assume that adversaries are not capable for MITM attack and therefore, Chronos receives timestamps from a trusted system call. This issue is further discussed in Section VII.

5. We consider that no entity is capable of modifying the implementation of the protocol. Hence, a malicious cloud provider cannot prepare a VM image that does not include the original implementation of Chronos. This issue is also explained in Section VII.

C. Threat Model

There can be different types of attacks related to the alteration of the system time, which are presented below:

- To hide the trace of any malicious activity using a cloud VM, a user can alter the current system time of the VM to any past or future time and later reset it to the actual time.
- A malicious user can collude with a dishonest employee of a CSP, or can take control of the host machine so that the timestamp of the VM and the host can be altered simultaneously before any malicious activity and reset to the original time afterwards.
- Beside the host, an attacker can compromise a CS to alter the timestamps of the VM, host, and CS simultaneously.
- A dishonest employee of a CSP can change the host machine's timestamp to frame an honest user.
- VMs, hosts, and CSs can fraudulently claim each others' valid timestamps as invalid, and vice versa.
- VMs, hosts, or CSs can deny any timestamp attestations and certifications.
- A certification statement, which also includes the attestation can be altered, removed, or reordered by a malicious CSP or an investigator to save an attacker or to frame an honest user.
- VMs, hosts, or CSs can deliberately introduce timing delays in the verification phase by packet dropping or jamming.

D. System Property

Based on the attacker's capability and possible attacks, a secure timestamp management system for clouds should ensure the following integrity and availability properties:

I1: A user cannot alter the timestamp of a VM whether acting alone or colluding with a compromised host without being detected by auditors.

I2: A CSP, colluding with a malicious investigator cannot alter the timestamp of the host or CS without being detected by auditors.

I3: A CSP, or an investigator cannot modify any attestation and certification statement nor can remove or reorder any certifications without being detected by auditors.

I4: A user, colluding with a CSP cannot alter the timestamp of the VM, host, and CS simultaneously without being detected by auditors.

I5: While acting alone, a VM, host, or CS cannot fraudulently claim the others timestamp as invalid without being detected by auditors.

I6: A CSP or user cannot repudiate any attestations and certifications.

A1: None of the entities can intentionally introduce any timing delay in the timestamp verification stage.

IV. The Chronos System

In Chronos, three entities – VM, host, and Chronos server (CS) participate in a timestamp verification protocol, where each entity verifies the timestamp of others. Later, information of the timestamp verification phase are stored securely using cryptographic schemes. Before beginning the verification cycle, VM and CS determine the Round Trip Time (RTT) with the host. Validity of a requestor's timestamp depends on the current timestamp of the verifier and the RTT values. In Chronos, the timestamp of one requestor is attested by two other entities and each attestation is later certified by an entity other than the verifier and requestor. Public key encryption and signature generation take place in all the communications to preserve the integrity of the scheme. At the end of each epoch, the proof publisher Brizo[2] prepares proofs of timestamp verification phase and makes the proofs publicly available. One host machine can make provision of multiple VMs and one CS can serve multiple host machines. One Brizo can be used to publish proofs collected from multiple Chronos servers.

A. Message Definitions

Notation: We use VM, H, and CS as the identity of the virtual machine, host, and Chronos server respectively. $Hash(M)$ is a collision resistant, one-way hash function, which produces a hash of a message M. The $S_K(M)$ function generates a signature of a message M using a secret key K. An attribute of a message is represented by $MessageIdentity.Attribute$. For concatenation of two messages, we use the symbol '|'. A tuple is encapsulated in '<>'.

[2]Brizo: comes from ancient Greek, referred as the patron Goddess of sailors, who sent prophetic dreams

Verification Request Message: Given $ReqID$ is the requestor's identity and T_{ReqID} is the timestamp of the requestor, a verification request message is defined as follows:

$$VR_{ReqID} =< S_{K_{ReqID}}(T_{ReqID}), T_{ReqID} >, \tag{1}$$

where $S_{K_{ReqID}}(T_{ReqID})$ is the requestor's signature on the requested timestamp using its private key K_{ReqID}.

Attestation Message: An attestation message A_{VerID_ReqID} is defined as follows:

$$A_{VerID\_ReqID} =< S_{K_{VerID}}(AI_{VerID\_ReqID}), AI_{VerID\_ReqID} >, \tag{2}$$

where the attested information (AI) is generated as follows:

$$AI_{VerID\_ReqID} =< Response(R), ATime, T_{ReqID}, ReqID, VerID > \tag{3}$$

Here, Response(R) is either true or false based on the verification result, $ATime$ is the attestation timestamp, T_{ReqID} is the requested timestamp, $ReqID$ is the requestor's identity, and $VerID$ is the verifier's identity.

Certification Message: A certification message C_{CertID_VerID} is defined as follows:

$$C_{CertID\_VerID} =< S_{K_{CertID}}(CI_{CertID\_VerID}), CI_{CertID\_VerID} > \tag{4}$$

The certified information CI_{CertID_VerID} is shown below:

$$CI_{CertID\_VerID} =< A_{VerID\_ReqID}, Response(R), CTime, VerID, CertID >, \tag{5}$$

where Response (R) is ether true or false depending on the validity of the attestation message. $CTime$ is the certification timestamp, $CertID$ is the certifier's identity, $VerID$ is the verifier's identity, and A_{VerID_ReqID} is the attestation message that is being certified.

Certification-Chain (CC): The Certification-Chain (CC) preserves the integrity of the certification messages, which is defined as follows:

$$[CC_{ReqID}]_{new} =< Hash([C_{CertID\_VerID}]_{new} | [CC_{ReqID}]_{prev}) > \tag{6}$$

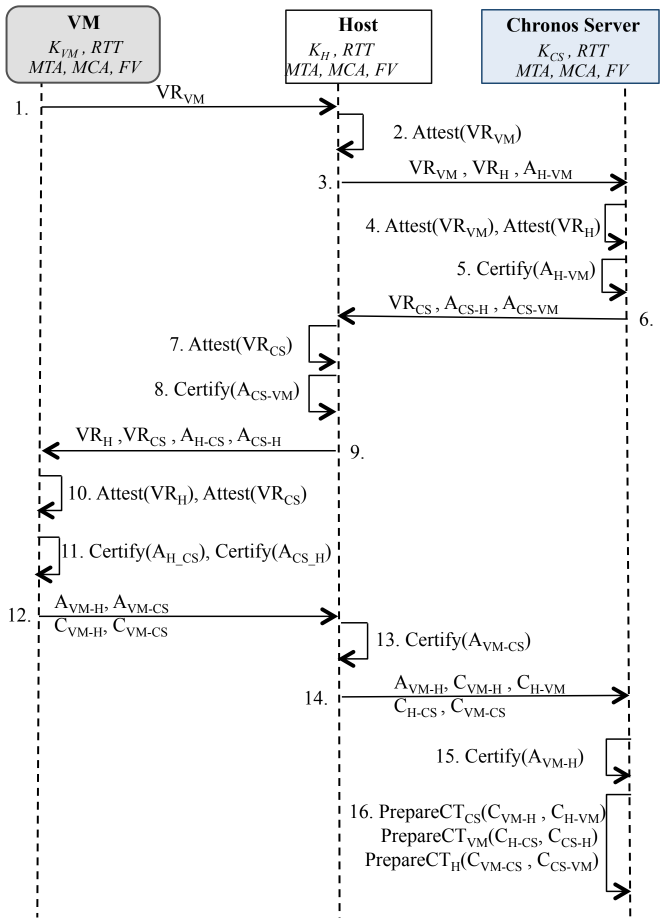
Here, $[CC_{ReqID}]_{prev}$ is the certification-chain, which was calculated for the previous certification message. If there is no previous certification message, a constant value can be used for the $[CC_{ReqID}]_{prev}$.

Certified-Timestamp (CT): A certification message and its associated certification-chain are stored in a persistent storage in the form of a Certified-Timestamp (CT), which is constituted of C_{CertID_VerID} and CC_{ReqID},

$$CT_{ReqID} =< C_{CertID\_VerID},\ CC_{ReqID} > \tag{7}$$

Proof of Time: If $[CC_{ReqID}]_N$ is the last certification-chain at the end of an epoch, the proof of time POT_{ReqID} for that epoch is generated as follows:

$$POT_{ReqID} =< S_{K_{CS}}([CC_{ReqID}]_N), [CC_{ReqID}]_N > \tag{8}$$



VM
K_{VM}, RTT
MTA, MCA, FV
Host
K_H, RTT
MTA, MCA, FV
Chronos Server
K_{CS}, RTT
MTA, MCA, FV
1. VR_{VM}
2. Attest(VR_{VM})
3. VR_{VM}, VR_H, $A_{H\text{-}VM}$
4. Attest(VR_{VM}), Attest(VR_H)
5. Certify($A_{H\text{-}VM}$)
6. VR_{CS}, $A_{CS\text{-}H}$, $A_{CS\text{-}VM}$
7. Attest(VR_{CS})
8. Certify($A_{CS\text{-}VM}$)
9. VR_H, VR_{CS}, $A_{H\text{-}CS}$, $A_{CS\text{-}H}$
10. Attest(VR_H), Attest(VR_{CS})
11. Certify(A_{H_CS}), Certify(A_{CS_H})
12. $A_{VM\text{-}H}$, $A_{VM\text{-}CS}$ $C_{VM\text{-}H}$, $C_{VM\text{-}CS}$
13. Certify($A_{VM\text{-}CS}$)
14. $A_{VM\text{-}H}$, $C_{VM\text{-}H}$, $C_{H\text{-}VM}$ $C_{H\text{-}CS}$, $C_{VM\text{-}CS}$
15. Certify($A_{VM\text{-}H}$)
16. PrepareCT$_{CS}$($C_{VM\text{-}H}$, $C_{H\text{-}VM}$)
PrepareCT$_{VM}$($C_{H\text{-}CS}$, $C_{CS\text{-}H}$)
PrepareCT$_H$($C_{VM\text{-}CS}$, $C_{CS\text{-}VM}$)

Fig. 1: Timestamp verification protocol

B. *Timestamp Verification Protocol*

We assume that before the verification phase is run, VM, host, and CS have setup their public and private keys (PK, K) and distributed the public keys. Additionally, VM and host setup RTT_{VM_H}; host and CS set up RTT_{H_CS}. The values of the two RTTs are calculated from the time difference between sending a *hello* message from one entity to another through a secure channel and receiving a response. Hence, computation time for signature generation and signature verification are included with the value of the $RTTs$. There are three other properties that are constant throughout each verification cycle: maximum tolerable timestamp error for attestation (MTA), maximum tolerable timestamp error for certification (MTC), and the frequency of the verification cycle being simulated (FV). We assume that all the entities are in the same time zone. Figure 1 shows the Chronos timestamp verification protocol and details of the protocol are described below:

1. The verification protocol starts with a VM sending verification request message VR_{VM} to the host machine.

2. Upon receiving a VR_{VM} message, the host first verifies the signature of the message to make sure that the message comes from a trusted VM. Signature verification is mandatory in all the communications of the protocol. Here, the host is the verifier and VM is the requestor; hence, the attested information is denoted as AI_{H_VM} and response of AI_{H_VM} message is determined as follows:

$$AI_{H\_VM}.R = \begin{cases} True, & |TN_H - (T_{VM} \\ & +RTT_{VM\_H})| \leq MTA \\ False, & Otherwise \end{cases} \quad (9)$$

Here, TN_H is the current timestamp of the host. Equation 9 determines the validity of the VM's timestamp with respect to TN_H and RTT_{VM_H}. Since it requires RTT_{VM_H} time to send a message from VM to host, TN_H must be close to $(T_{VM} + RTT_{VM\_H})$ for the response $AI_{H\_VM}.R$ to be true; otherwise, the response is false. A false response does not terminate the timestamp verification protocol; in Chronos, a false response is recorded the same way that a true response is recorded. Once the response is determined, the host creates the attestation message A_{H_VM} using Equations 2 and 3.

3. The host issues a timestamp verification request VR_H to the CS. It also sends the VR_{VM} message received from the VM and the attestation message A_{H_VM} to the CS.

4. The CS attests the requested timestamps of the VM and the host, and builds attestation messages A_{CS_VM} and A_{CS_H} respectively. The response of AI_{CS_VM} is determined as follows:

$$AI_{CS\_VM}.R = \begin{cases} True, & |TN_{CS} - (T_{VM}+ \\ & RTT_{VM\_H} + RTT_{H\_CS})| \leq MTA \\ False, & Otherwise \end{cases} \quad (10)$$

Here, TN_{CS} is the current timestamp of the CS. The CS receives the timestamp of the VM, T_{VM}, from the host. Therefore, if the difference between TN_{CS} and $(T_{VM} + RTT_{VM\_H} + RTT_{H\_CS})$ is within the range of MTA, the response is true, and false otherwise. Next, response of the AI_{CS_H} message is determined as follows:

$$AI_{CS\_H}.R = \begin{cases} True, & |TN_{CS} - (T_H + RTT_{H\_CS})| \leq MTA \\ False, & Otherwise \end{cases} \quad (11)$$

In this case, timestamp of the host, T_H, is directly sent from the host to the CS. Hence, for a valid T_H, the current timestamp of the CS, TN_{CS}, should be close to $(T_H + RTT_{H\_CS})$.

5. To detect a fake attestation by a host, the CS checks and certifies the attestation message A_{H_VM}. If the CS attests VR_H as invalid, the CS also certifies the attestation message A_{H_VM} as false. Otherwise, the CS goes for further checking. Since the current time of host in Equation 9, TN_H, is actually the verification time $AI_{H\_VM}.ATime$, $d1 = |AI_{H\_VM}.ATime - (T_{VM} + RTT_{VM\_H})| \leq MTA$ should be held true when the response $AI_{H\_VM}.R$ is true.

The host creates the VR_H just after verifying the VM's timestamp. Hence, the requested timestamp of the host, T_H, should be close to the verification time $AI_{H\_VM}.ATime$. We denote the difference of these two timestamps as $d2 = |T_H - A_{H\_VM}.ATime|$. The CS now determines the response of CI_{CS_H} according to Algorithm 1, where the arguments of the DetermineCertifyResponse method are AI_{H_VM}, $d1$, and $d2$. Using the response value, the CS creates the certification message C_{CS_H} according to Equations 4 and 5.

6. The CS sends attestation messages A_{CS_H}, A_{CS_VM}, and the verification request message VR_{CS} to the host.

Algorithm 1 Determining response of certification

```
1: DetermineCertifyResponse(AI, d1, d2)
2:   if ($AI.response = True$)
3:       if (($d1 + d2$) $\leq MTC$) $response = True$
4:       else $response = False$
5:   else
6:       if (($d1 + d2$)$>MTC$)) $response = True$
7:       else $response = False$
8: return response
```

7. In this step, the host attests the requested timestamp of the CS, T_{CS}. If the requested timestamp of the CS is valid, the current timestamp of host, TN_H, should be close to $(T_{CS} + RTT_{H\_CS})$. Therefore, the response of the attestation information, AI_{H_CS}, is determined as follows:

$$AI_{H\_CS}.R = \begin{cases} True, & |TN_H - (T_{CS} + RTT_{H\_CS})| \leq MTA \\ False, & Otherwise \end{cases} \quad (12)$$

After determining the response, the host creates the attestation message A_{H_CS} according to Equations 3 and 2.

8. In this step, the host certifies the attestation message A_{CS_VM}. If the host attests VR_{CS} as invalid, the response of the certification $CI_{H\_CS}.R$ becomes false; otherwise, the host goes for further checking. According to Equation 10, $d1 = |AI_{CS\_VM}.ATime - (T_{VM} + RTT_{VM\_H} + RTT_{H\_CS})| \leq MTA$ as $AI_{CS\_VM}.ATime = TN_{CS}$. The CS creates the VR_{CS} just after verifying the VM's timestamp. Hence, $d2 = |T_{CS} - AI_{CS\_VM}.ATime|$ should be small. Using AI_{CS_VM}, $d1$, and $d2$ as the arguments in Algorithm 1, the host identifies the response of the CI_{H_CS} and creates the certification message C_{H_CS} according to Equations 4 and 5.

9. The host sends attestation messages A_{H_CS} and A_{CS_H}, and verification request messages VR_H and VR_{CS} to the VM.

10. In this step, the VM first attests the requested timestamp of host, T_H. Though the message VR_H is directly sent from the host to the VM, the message was actually created in step 3. After that, the message was attested by the CS and the host received the attestation results back from the CS. Hence, while comparing the requested timestamp of the host, T_H, with the current timestamp of the VM, TN_{VM}, we need to consider two RTT values: RTT_{VM_H} and RTT_{H_CS}. For a valid T_H, current timestamp of the VM should be close to $(T_H + RTT_{VM\_H} + RTT_{H\_CS})$. Accordingly, verification response of AI_{VM_H} is determined as follows:

$$AI_{VM\_H}.R = \begin{cases} True, & |TN_{VM} - (T_H + RTT_{VM\_H} \\ & +RTT_{H\_CS})| \leq MTA \\ False, & Otherwise \end{cases} \quad (13)$$

Next, the VM attests the timestamp of the CS. The verification request message VR_{CS} is sent from the CS to the VM via the host. Hence, the two RTT values are involved in determining the validity of the CS's timestamp. Therefore, the verification response of AI_{VM_CS} can be determined by the following Equation:

$$AI_{VM\_CS}.R = \begin{cases} True, & |TN_{VM} - (T_{CS} + RTT_{VM\_H} \\ & +RTT_{H\_CS})| \leq MTA \\ False, & Otherwise \end{cases} \quad (14)$$

11. The VM certifies two attestation messages A_{H_CS} and A_{CS_H}. The response of the certification information CI_{VM_H} is determined as follows:

First, if the VM attests VR_H as invalid, it also certifies the A_{H_CS} as invalid. Moreover, since $TN_H = AI_{H\_CS}.ATime$, $d1 = |AI_{H\_CS}.ATime - (T_{CS} + RTT_{H\_CS})| \leq MTA$, when A_{H_CS} is valid (Equation 12). Additionally, $d2 = |T_H - AI_{H\_CS}.ATime|$ should be small. Now, using AI_{H_CS}, $d1$, and $d2$ as the arguments in the Algorithm 1, the VM identifies the response of the CI_{VM_H}. Next, the response of certification information CI_{VM_CS} is determined as follows:

The VM certifies A_{CS_H} as invalid, if it attests the VR_{CS} as false. Since $TN_{CS} = AI_{CS\_H}.ATime$, if the timestamp of CS is valid, $d1 = |AI_{CS\_H}.ATime - (T_H + RTT_{H\_CS})| \leq MTA$ (Equation 11). Here, $d2 = |T_{CS} - AI_{CS\_H}.ATime|$ should be very small. Using AI_{CS_H}, $d1$, and $d2$ as the arguments in the Algorithm 1, the VM identifies the response of the CI_{VM_CS}.

12. The VM sends attestation messages A_{VM_H} and A_{VM_CS}, and certification messages C_{VM_H} and C_{VM_CS} to the host.

13. In this step, the host certifies A_{VM_CS}. If the response of the attestation message A_{H_VM} was false in the step 2 of the protocol, the host certifies A_{VM_CS} as invalid. When a honest VM attests T_{CS} as valid, according to Equation 14, $d1 = |AI_{VM\_CS}.ATime - (T_{CS} + RTT_{VM\_H} + RTT_{H\_CS})| \leq MTA$ as $AI_{VM\_CS}.ATime = TN_{VM}$. Now, between step 10 (when the VM attest VR_{CS}) and step 1 (when the VM first issues the VR_{VM}), we observe two rounds of message transfers between the VM and the host, and the CS and the host. Hence, if there is no unwanted network delay, $d2 = |T_{VM} + 2RTT_{VM\_H} + 2RTT_{H\_CS} - AI_{VM\_CS}.ATime|$ should be very small. Using AI_{VM_CS}, $d1$, and $d2$ as the arguments in the Algorithm 1, the CS determines the response of the CI_{H_VM}.

14. The host sends the attestation message A_{VM_H} and certification messages C_{VM_H}, C_{H_VM}, C_{H_CS}, and C_{VM_CS} to the CS.

15. In this step, the CS certifies A_{VM_H}. The CS certifies A_{VM_H} as invalid if it attested VR_{VM} as invalid in step 4. Otherwise, the CS will go for further checking. Since $AI_{VM\_H}.ATime = TN_{VM}$, $d1 = |AI_{VM\_H}.ATime - (T_H + RTT_{VM\_H} + RTT_{H\_CS})| \leq MTA$ when AI_{VM_H} is valid (Equation 13). As discussed in step 13, $d2 = |T_{VM} + 2RTT_{VM\_H} + 2RTT_{H\_CS} - A_{VM\_H}.ATime|$ should be very small if the VM is honest. Using AI_{VM_H}, $d1$, and $d2$ as the arguments in the Algorithm 1, the CS determines the response of the CI_{CS_VM}.

16. In the last step of the protocol, the CS stores all the certification information in a persistent storage. To maintain the integrity of the certification records, the CS uses Equations 6 and 7 to generate a tuple for the storage. For example, the PrepareCT$_{\text{VM}}$ method generates a CC_{VM} from C_{H_CS} according to Equation 6, creates CT_{VM} by following Equation 7, and stores the CT_{VM} in the storage. By following the same approach, this method creates and stores another CT_{VM} generated from C_{CS_H}. The others methods, PrepareCT$_{\text{CS}}$ and PrepareCT$_{\text{H}}$ similarly generate and store CT_{CS} and CT_H respectively.

Prepare and Publish Proofs: Chronos server stores the attestations and certifications, which are prepared by the VM, host, and CS. However, a dishonest cloud provider can alter the traces stored within the CS. Hence, the CS creates POT_{CS}, POT_{VM}, and POT_H using Equation 8 and all of the proofs of traces will be published publicly by Brizo at regular intervals. Later, an auditor can use the published proofs of traces to detect alteration of attestations and certifications.

C. *Verification by Auditor*

Having Chronos integrated with the cloud, an investigator can gather all the certified timestamps of the time range that is under inspection and present to the auditor. To verify the integrity of the certified timestamps, CT_{VM}, CT_H, and CT_{CS}, provided by the investigator, an auditor collects the proof of time POT_{VM}, POT_H, and POT_{CS}.

Integrity Verification: The auditor needs to check whether the certified timestamps provided by the investigator have been tampered with or not. A malicious cloud provider or dishonest investigator can alter the attestation or certification information. The auditor should be able to detect any alteration of attestations or certifications during the verification process.

Here, we consider the certified timestamps of only one entity to present the integrity verification phase. The verification process that is presented in Figure 2 starts with verifying the signature of the CS on the proof of time (POT_{ReqID}). After satisfying with the signature, the auditor extracts the certification messages, $C_0..C_N$, from certified timestamps $[CT_{ReqID}]_0...[CT_{ReqID}]_N$ and sorts them according to the certification timestamp ($CTime$). Next, applying Equation 6 on all the certifications chronologically provides the last certification-chain value, $[CC_{ReqID}]_G$. This value should match with the certification-chain $[CC_{ReqID}]_N$ of Equation 8 to prove that the certified timestamps have not been tampered with.

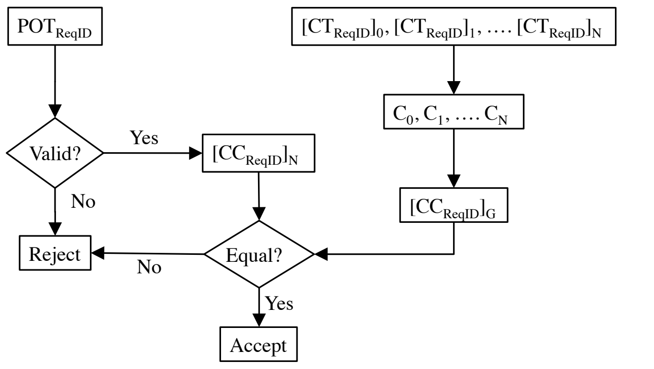


Fig. 2: Process flow for integrity verification

V. Security Analysis

In this section, we discuss how Chronos ensures the security properties mentioned in Section III-D.

Claim 1: *Chronos ensures that a user cannot alter a VM's timestamp whether acting alone or colluding with a host without being detected.* (Property I1)

Justification. According to Equations 9 and 10, any changes in the timestamp of a VM, which is greater than MTA can be detected by a host and CS respectively. A colluding host can maintain the same fake time as the VM and attest the VM's false timestamp as valid. However, as both the VM and the host maintain false time, the CS can detect this alteration according to Equations 10 and 11. Additionally, the certification stage can detect fake attestation by two colluding entities. Hence, as we discussed in step 5 and 15 of the Chronos protocol, even if the host and the VM attest each other's fake timestamps as valid, the CS will certify the attestations of the VM and host (A_{H_VM} and A_{VM_H}) as invalid. Thus, because of the presence of an honest CS, users cannot alter the VM's timestamp even after colluding with a host without being detected by an auditor.

Claim 2: *Chronos ensures that a CSP, colluding with a malicious investigator cannot alter the timestamp of the host or CS without being detected.* (Property I2)

Justification. If the host machine's timestamp is altered, CSs and VMs can detect this modification while attesting a host's timestamp request, VR_H, using Equations 11 and 13. However, if the CS also gets compromised, adversaries can maintain the same false timestamp in the CS and the host machine. Such a CS and host will attest each other's timestamp as valid, and both will attest the VM's timestamp as invalid. The malicious behavior of the host and the CS can be detected from the response of the attestations A_{VM_H} and A_{VM_CS}, when a VM verifies the timestamp of a host and a CS. In this scenario, the VM will also certify the attestation A_{H_CS} and A_{CS_H} as invalid according to our discussion about step 11 of the protocol. On the other hand, timestamp requests and attestations of an honest VM will be certified as false by a compromised host and CS. In this case, the auditor can decide about the honesty of a user or CSP based on a collection of attestations and certifications issued by all the VMs hosted on the suspected host machine. As VM owners are honest in this case, attestations and certifications issued by all of the VMs will indicate that timestamps of host and CS are invalid.

Claim 3 *Chronos detects modification, removing, and reordering of attestations and certifications by any entity (user, CSP, and investigator)* (Property I3)

Justification: When a CSP or an investigator alters any of the attestation or certification information, this will change the certified timestamps. As we illustrated in Figure 2, when an adversary alters, removed, or reordered any of the certified timestamps CT_{Req}, the $[CC_{ReqID}]_G$ will not match with the $[CC_{ReqID}]_N$ according to the hash-chain scheme. However, the $[CC_{ReqID}]_N$ is signed by the CS and published to the Internet. Hence, using the proof of time, POT_{ReqID}, any modification, removal, and reordering of certified timestamps can be detected by the auditor.

Claim 4 *Chronos ensures that a user, colluding with a CSP cannot manage to alter the timestamp of the VM, host, and CS simultaneously without being detected.* (Property I4)

Justification: Let us assume that a CSP and a user were honest until T1 timestamp, i.e., until T1, host, CS, and VM generated trusted attestations and certifications. Later, the CSP and the user colluded and set the timestamp of the VM, host, and CS to T2, where $T2<T1$. Certifications for T2 was added to the certification-chain CC, where the last certification in the chain had timestamp T1. During verification, the auditor sorts the certifications by certification timestamps. Therefore, the auditor generated $[CC_{ReqID}]_G$ will not match with $[CC_{ReqID}]_N$. Without violating the chain, a user or CSP cannot place $[CT_{ReqID}]_{T2}$ before $[CT_{ReqID}]_{T1}$.

Now, let us assume that a CSP and a user set a false timestamp T3 in the host, CS, and VM, where $T3>T1$. Later, they reset the timestamp to actual timestamp T4, where $T4<T3$. Without violating the certification-chain a user or CSP cannot place $[CT_{ReqID}]_{T4}$ before $[CT_{ReqID}]_{T3}$. If the VM or host does not reset their time to the actual timestamp, this can be trivially detected from the timestamp associated with the latest certification and attestations.

A user and a CSP could also be dishonest from the very beginning and maintained a fake timestamp, which is less than the actual timestamp. Later, they can reset the timestamp to actual time this alteration cannot be detected by using the hash-chain. For such an attack, auditors can determine the trustworthiness of the timestamp from the attestations and certifications of other honest, tenant VMs served by the same CS and host. The honest VMs will not attest and certify the timestamp of host and CS as valid. Therefore, if the majority of the VMs (greater than 50%) detect hosts and CS timestamp as false, the auditor determines the CSP and the colluding user as dishonest. The required percentage to determine the majority can vary with the size and type of clouds.

Claim 5 *Chronos ensures that while acting alone, a VM, host, or CS cannot fraudulently claim the others timestamp as invalid without being detected. (Property I5)*

Justification: Because of the certification stage, the proposed system ensures that a VM, host, or CS cannot fraudulently claim the others timestamp as invalid while acting alone. A verifier with a fake timestamp can attest an honest requestor's timestamp as false. However, such an attestation will be certified as false by the honest certifier. When one entity attests two of the other entities timestamp as invalid, both of the attestations will be certified as invalid by the honest entities. For example, when a malicious VM attests the timestamp of a trusted host as false, this attestation will be certified as false by an honest CS. Similarly, the trusted timestamp of the CS, which is attested as false by the malicious VM will be certified as invalid by the honest host. Hence, when the VM, host, and CS are acting alone, they cannot fraudulently claim each other's timestamp as invalid.

Claim 6 *Chronos ensures non-repudiation of attestations and certifications by users and CSPs.* (Property I6)

Justification: After attesting a timestamp or certifying an attestation, users or CSPs cannot repudiate the attestations and certifications, since these are signed by a user's and CSP's private key. Nobody, other than the owner of a private key can use the key to sign the attestation and certification information. Hence, if an attestation message has the AI with a valid signature, a CSP or user cannot repudiate the attestation. Similarly non-repudiation of certification is also ensured.

Claim 7 *Chronos identifies any unwanted timing delay introduced by a VM, host, or CS at the time of certification.* (Property A1)

Justification: We prove this claim by considering the host as the responsible entity for introducing timing delay. The host forwards the verification request VR_{VM} to the CS after attesting the VM's timestamp. Hence, the attestation time, $AI_{H\_VM}.ATime$, and the timestamp that the host wants to verify, T_H, should be very close to each other. As we discuss in step 5 of the protocol, while certifying A_{H_VM}, the CS checks the temporal locality of the attestation time $AI_{H\_VM}.ATime$ and requested timestamp T_H, and can detect any timing delay.

The verification request VR_{CS} is sent from a CS to a VM via the host. Hence, a similar kind of timing delay attack can be launched by a malicious host. If the host introduces the delay, it will be identified by a VM while certifying A_{H_CS}. In step 11 of the protocol, the VM checks the temporal locality of the attestation time $A_{H\_CS}.ATime$ and the timestamp requested by the host, T_H. Hence, it is not possible for a host to introduce timing delay in CS's verification request without being identified by a VM. We can show similar proofs considering the VM and the CS as the attacking entity.

VI. Implementation and Evaluation

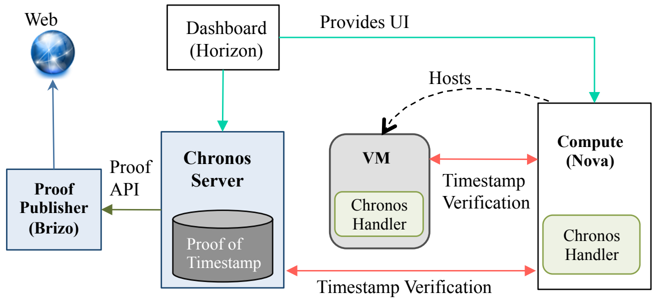


Fig. 3: Chronos for OpenStack-based clouds

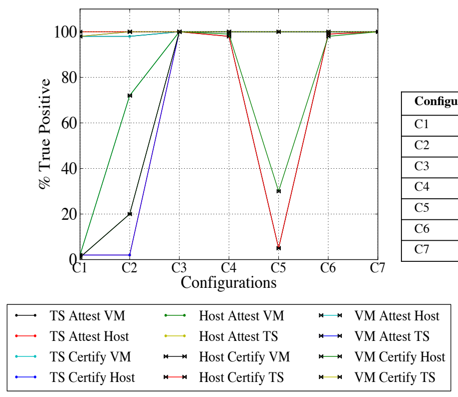
A. Implementation

We integrate Chronos with an OpenStack-based private cloud. Figure 3 presents the integration of the Chronos server and the proof publisher (Brizo) with the OpenStack cloud. In OpenStack, VMs are hosted on the Nova compute node and both are enhanced with a new component — Chronos Handler that participates in the timestamp verification process. We augment the OpenStack Dashboard (Horizon) module to display and collect the attestation and certification messages. Using the proof collection API provided by the Chronos server, Brizo collects the proofs of timestamps, POTs, at the end of each epoch and publishes the proofs to the Internet.

System Configuration: Our cloud is running on one cloud controller and one Nova compute node. The Nova compute and Chronos server have four Intel(R) Xeon(R) CPU E31220 (3.10GHz) processors with 32GB RAM and 8MB cache. For our experiment, we integrate Brizo with the Chronos server. However, for large-scale deployment, a dedicate machine for Brizo would be preferable. We used 20 m1.small instances, running the Ubuntu 12.04.4 LTS operating system as VMs. The hardware configurations of node controller allowed us to run a maximum 20 m1.small instances. The CPUs of the VMs are single core Intel Xeon E312xx (Sandy Bridge) 3.10 GHz with 2GB RAM and 4MB cache. We used OpenJDK (version:1.6.0 27) to implement the Chronos protocol. We used RSA (2048 bit) for signature generation and SHA-2(SHA-256) hash function for hashing.

B. Evaluation

Identifying Optimum MTA and MTC: The two threshold values: maximum tolerable timestamp error for attestation (MTA) and for certification (MTC) play vital roles in the timestamp verification protocol. We identified the optimum value for these two properties based on the true positive (TP) rate for attestations and certifications. True positive means all of the entities are honest and according to Chronos, all of the entities attest and certify each other as honest. Since MTA and MTC depend on the RTT, we chose these two properties as different factors of the RTT and measured the true positive rate. To identify the TP rate, we run Chronos in all of the 20 VMs simultaneously, where each VM runs 1,000 cycles of the protocol. Hence, cumulatively the TP rate was identified from 20,000 cycles of the protocol. Figure 4 justifies the MTA and MTC identification, where the X-axis shows different configurations and Y-axis represents the TP rate of attestations and certifications. Since increasing the threshold values not only increases the TP rate but also increases the false positive (dishonest entity detected as honest) rate, our goal was to find the lowest MTA and MTC with the highest true positive rate. We started with the configuration C7 and gradually decreased the value of MTA and MTC. At configuration C5, the rate of TP dropped for certifications. Hence, we increased the MTC value again and gradually decreased the value of MTA. Later, we noticed 100% TP rate for the configuration C3 ($MTA = 0.8 * RTT$ and $MTC = 1.5 * RTT$). When we decreased the MTA and MTC values below the configuration of C3, the TP rate decreased again. Hence, we determined configuration C3 as the optimum configuration.



Configuration	MTA	MTC
C1	0.5*RTT	1.5*RTT
C2	0.6*RTT	1.5*RTT
C3	0.8*RTT	1.5*RTT
C4	RTT	1.5*RTT
C5	RTT	RTT
C6	1.5*RTT	1.5*RTT
C7	2*RTT	2*RTT

Fig. 4: Identifying MTA and MTC

Identifying Frequency of Verification (FV): Long time durations between the timestamp verification cycles gives a sufficient window for timestamp alteration attack, which will remain undetected. Conversely, running the verification protocol very frequently introduces higher system overhead. Hence, our target is to run the verification protocol as frequently as possible, which keeps the system overhead in a tolerable range. As illustrated in Figure 5, the system overhead of the host increases while increasing the number of VMs and the frequency of verification protocol. We measured the system overhead from the CPU performance information of SysBench [33]. Using the least square optimization method, we approximated the relation between system overheads, number of VMs (nVM), and FV for 10 VMs, which is defined as follows:

$$\%Overhead = nVM * 0.05 \ + nVM * 0.4 * exp(-0.3 * VF) \quad (15)$$

According to Equation 15, the lowest and highest residual sum of squares (RSS) found for 1 to 10 VMs are 0.02 and 6.18 respectively. Later, we tested this hypothesis for 20 VMs and the RSS value was 0.89, which was inside the previously found range. From the Figure 5, we also notice that the hypothesis derived from 1 to 10 VMs is very close to the actual results for 20 VMs. Hence, from Equation 15, a CSP can determine the value of FV from the number of VM's running on a host and a performance overhead tolerable to CSP. For example, for 20 VMs with 1% expected system overhead of host machine, a CSP can set the FV value from 30 to 60 seconds.

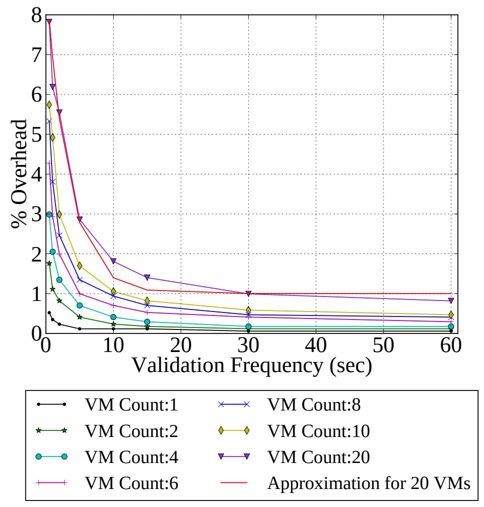


Fig. 5: Identifying Frequency of Verification (FV)

Stability of the System: If the protocol fails frequently, attackers may get sufficiently large window to attack on the timestamp. To measure the stability of the Chronos, we kept Chronos running in all of the 20 VMs for 7 days with verification frequency 5 minutes. Among the 20 VMs, we randomly selected 8 VMs to change their system time. We set the system time of 4 VMs 5 minutes ahead of the original time and 4 VMs 5 minutes past the original time. System time of the host and the CS were set to the original time. We calculated the true positive (TP) and true negative (TN) rate of each type of attestations and certifications, and the success rate of the completion of the verification cycles. We found that 94.67% of the timestamp verification cycles were completed successfully. The rate of TP was 100% for all types of attestations and certifications and the rate of TN varied between 94.75% and 100%. Since, the system time of the host and the CS were set to original, there were no true negative rate when the host and the CS attest or certify each other. The experimental results are presented in Table I.

Action	% True Positive	% True Negative
VM Attest Host	100	94.75
VM Attest CS	100	94.75
VM Certify Host	100	94.75
VM Certify CS	100	94.75
Host Attest VM	100	100
Host Attest CS	100	–
Host Certify VM	100	100
Host Certify CS	100	–
CS Attest VM	100	100
CS Attest Host	100	–
CS Certify VM	100	100
CS Certify Host	100	–

TABLE I: Results of system stability while running Chronos for 7 days

VII. Discussion

In this section, we examine several critical adversarial scenarios and propose some possible alternative solutions to avoid some of the adversarial cases.

Man in the Middle (MITM) Attack: In Chronos, all the entities receive the current timestamp through a system call. However, we investigate that using *ptrace* [34] system call, a user with root privilege can trace the Chronos process and change the return value of *SYS_time* call by overriding the value of registers. We argue that allowing ptrace to capture or change the system call's return value will make many of existing security schemes vulnerable, such as SSH session hijacking and arbitrary code injection attacks [35]. Since ptrace is not commonly used by non-developers and non-admins, system builders should be allowed to disable this system call by using SELinux and Yama security modules [36], [37], [38].

MITM attack is also possible by modifying the memory by changing /dev/mem. This is a complex attack that involves finding where the user-level process is located and modifying it on-the-fly. System calls can also be altered by compromising the operating system's (OS) kernel. Security of OS is orthogonal to the focus of our work and it can be achieved by using a Trusted Platform Module (TPM) [39], [40], [41].

Untrusted Chronos: We consider that the CSP or users cannot modify the proposed timestamp verification protocol. However, one can argue that a dishonest CSP can integrate a different version of Chronos with the VM image, which does not detect the incorrect timestamp of host machine or CS. Moreover, Chronos running in the host machine or the CS can also be altered. We propose that the cryptographic hash of the original Chronos will be available on the Internet through a trusted website. Users can compare the hash of Chronos that comes with their VM image with the one that is available on the website. On the other hand, Chronos running in host machines and CSs can be attested by TPM to ensure that original implementation of the protocol has not been compromised.

New Attack Surface: The communication channel between VM and host can be considered as a new attack surface. Exploiting this new communication channel, an adversary can attack the hypervisor as the hypervisor runs in the host machine. However, since we transfer messages through a secure channel using message encryption and signature, it is not possible for an intruder to come in the middle of the communication to take control over the communication channel.

VIII. Conclusion and Future Work

Alteration of the system time of a VM or host machine provides wrong information about the VM's activity. Adversaries can find this attack attractive to hide the trace of their malicious activities or to provide a fake alibi. In this paper, we proposed Chronos, a secure and trustworthy timestamp management system to detect this kind of malicious behavior of cloud users, where cloud service providers can also be dishonest. We integrate a prototype of Chronos with an OpenStack-based private cloud platform with minimal effort. The prototype runs with a high degree of stability. Our experimental results suggests that Chronos can be integrated with existing cloud platforms with very low system overhead while providing a high degree of trustworthiness of the timestamp of the host and VMs. We believe, Chronos will ensure reliable forensics investigation in clouds by providing trusted timestamps of VMs and cloud hosts.

Future work will be focused on evaluating the performance of Chronos on a larger scale cloud environment with multiple hosts and CSs. We will enhance the protocol to handle multiple time zones of the entities. At present, Chronos cannot ensure trustworthy timestamps of a VM that is running inside another VM. The first level VM participates in the protocol, but the second level VM does not. Therefore, we plan to extend Chronos to cope up with multi-level nested VM environments.

Acknowledgment

This research was supported by the National Science Foundation under the CAREER Award CNS-1351038.

References

[1] B. Deeter and K. Shen, "BVP Cloud Computing Index Crosses the $100 Billion Market Milestone," http://goo.gl/mEuEi4, 2013.

[2] Gartner, "Gartner says that consumers will store more than a third of their digital content in the cloud by 2016," http://goo.gl/39a0y, 2012.

[3] IDC, "U.S. Public IT Cloud Services Revenue Projected to Reach $43.2 Billion in 2016," http://goo.gl/nXL4t3, 2012.

[4] INPUT, "Evolution of the cloud: The future of cloud computing in government," http://goo.gl/KrKexK, 2009.

[5] Market Research Media, "Global cloud computing market forecast 2015-2020," http://www.marketresearchmedia.com/?p=839.

[6] Infosecurity-magazine, "Ddos-ers launch attacks from amazon ec2," http://goo.gl/vrXrHE, July 2014.

[7] The Register, "Amazon cloud hosts nasty banking trojan," http://goo.gl/xGNkNO, 2011.

[8] Dist. Court, SD Texas, "Quantlab technologies ltd. v. godlevsky," Civil Action No. 4: 09-cv-4039, 2014.

[9] www.bbc.com, "Lostprophets' Ian Watkins: 'Tech savvy' web haul," http://goo.gl/C8FVnC, December 2013.

[10] S. Thorpe and I. Ray, "File timestamps for digital cloud investigations," *Journal of Information Assurance & Security*, vol. 6, no. 6, 2011.

[11] E. Casey, "Error, uncertainty, and loss in digital evidence," *International Journal of Digital Evidence*, vol. 1, no. 2, pp. 1–45, 2002.

[12] OpenStack, "Open source software for building private and public clouds." http://www.openstack.org/, 2012.

[13] P. Gladyshev and A. Patel, "Formalising event time bounding in digital investigations," *International Journal of Digital Evidence*, vol. 4, no. 2, pp. 1–14, 2005.

[14] S. Thorpe and I. Ray, "Detecting temporal inconsistency in virtual machine activity timelines." *JIAS*, vol. 7, no. 1, 2012.

[15] L. Lamport, "Time, clocks, and the ordering of events in a distributed system," *Communications of the ACM*, vol. 21, no. 7, pp. 558–565, 1978.

[16] X. Chen, J. Li, J. Ma, Q. Tang, and W. Lou, "New algorithms for secure outsourcing of modular exponentiations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 25, no. 9, pp. 2386–2396, 2014.

[17] C. Erway, A. Küpçü, C. Papamanthou, and R. Tamassia, "Dynamic provable data possession," in *CCS*. ACM, 2009, pp. 213–222.

[18] X. Lei, X. Liao, T. Huang, H. Li, and C. Hu, "Outsourcing large matrix inversion computation to a public cloud," *IEEE Transactions on Cloud Computing*, vol. 1, no. 1, pp. 1–1, 2013.

[19] K. Y. Oktay, M. Gomathisankaran, M. Kantarcioglu, S. Mehrotra, and A. Singhal, "Towards data confidentiality and a vulnerability analysis framework for cloud computing," in *Secure Cloud Computing*. Springer, 2014, pp. 213–238.

[20] Z. Xu, C. Wang, K. Ren, L. Wang, and B. Zhang, "Proof-carrying cloud computation: The case of convex optimization," *IEEE Transactions on Information Forensics and Security*, vol. 9, no. 11, pp. 1790–1803, 2014.

[21] S. Zawoad, A. K. Dutta, and R. Hasan, "SecLaaS: Secure logging-as-a-service for cloud forensics," in *ASIACCS*. ACM, 2013, pp. 219–230.

[22] S. Ganeriwal, S. Čapkun, C.-C. Han, and M. B. Srivastava, "Secure time synchronization service for sensor networks," in *WiSec*. ACM, 2005, pp. 97–106.

[23] P. Maniatis and M. Baker, "Secure history preservation through timeline entanglement," in *USENIX Security Symposium*, 2002, pp. 297–312.

[24] S. Thorpe, I. Ray, I. Ray, and T. Grandison, "A formal temporal log data model for the global synchronized virtual machine environment," *JIAS*, vol. 6, no. 2, 2011.

[25] B. Schatz, G. Mohay, and A. Clark, "A correlation method for establishing provenance of timestamps in digital evidence," *Digital investigation*, vol. 3, pp. 98–107, 2006.

[26] M. W. Stevens, "Unification of relative time frames for digital forensics," *Digital Investigation*, vol. 1, no. 3, pp. 225–239, 2004.

[27] A. Boukerche and D. Turgut, "Secure time synchronization protocols for wireless sensor networks," *Wireless Communications, IEEE*, vol. 14, no. 5, pp. 64–69, 2007.

[28] K. Sun, P. Ning, and C. Wang, "TinySeRSync: Secure and resilient time synchronization in wireless sensor networks," in *CCS*. ACM, 2006, pp. 264–277.

[29] D. Bayer, S. Haber, and W. S. Stornetta, "Improving the efficiency and reliability of digital time-stamping," in *Sequences II*. Springer, 1993, pp. 329–334.

[30] A. Bonnecaze, P. Liardet, A. Gabillon, and K. Blibech, "Secure time-stamping schemes: A distributed point of view," in *Annales des Télécommunications*, vol. 61. Springer, 2006, pp. 662–681.

[31] A. Buldas, P. Laud, H. Lipmaa, and J. Villemson, "Time-stamping with binary linking schemes," in *CRYPTO*. Springer, 1998, pp. 486–501.

[32] S. HABER, "How to time-stamp a digital document," *Journal of Cryptology*, vol. 3, no. 2, pp. 99–111, 1991.

[33] SysBench, "Sysbench: a system performance benchmark," http://sysbench.sourceforge.net/.

[34] linux.die.net, "ptrace(2) - Linux man page," http://linux.die.net/man/2/ptrace.

[35] www.kernel.org, "YAMA," https://goo.gl/hsqi9E.

[36] K. Cook, "Security: Yama LSM," http://lwn.net/Articles/393012/, 2010.

[37] J. Corbet, "SELinuxDenyPtrace and security by default," http://lwn.net/Articles/491440/, April 2012.

[38] Fedoraproject.org, "Features / SELinuxDenyPtrace," http://goo.gl/gwdFxA.

[39] F. Krautheim, D. Phatak, and A. Sherman, "Introducing the trusted virtual environment module: a new mechanism for rooting trust in cloud computing," *Trust and Trustworthy Computing*, pp. 211–227, 2010.

[40] N. Santos, K. Gummadi, and R. Rodrigues, "Towards trusted cloud computing," in *HotCloud*. USENIX, 2009.

[41] Z. Shen and Q. Tong, "The security of cloud computing system enabled by trusted computing technology," in *ICSPS*, vol. 2. IEEE, 2010, pp. V2–11.