An Intrusion and Fault Tolerant Forensic Storage for a SIEM System

Muhammad Afzaal, Cesario Di Sarno, Salvatore D'Antonio, Luigi Romano
Department of Technology
University of Naples "Parthenope"
Naples, Italy
muhammad.afzaal,cesario.disarno,salvatore.dantonio,luigi.romano@uniparthenope.it

***Abstract*—Current Security Information and Events Management (SIEM) solutions lack a data storage facility which is secure enough - i.e. stored events related to security incidents cannot be forged and are always available - that it can be used for forensic purposes. Forensic storage used by current SIEM solutions uses traditional RSA algorithm to sign the security events. In this paper we have analyzed the limits of current forensic storages, and we have proposed an architecture for forensic storage, implementing a threshold-based variant of the RSA algorithm, that outperforms state of the art SIEM solutions in terms of intrusion- and fault-tolerance. We show by experiments that our forensic storage works correctly even in the presence of cyber-attacks, although with a performance penalty. We also conduct an experimental campaign to evaluate the performance cost of the proposed scheme as a function of the threshold.**

***Keywords*-Forensic Storage; Threshold Cryptography; Critical Infrastructure Protection; Fault- and Intrusion-Tolerant Architecture;**

I. Rationale and Contribution

Nowadays, the widespread use of Information Technology (IT) has led to growth in the number of attacks to which the systems are exposed [1] [2] [3]. The consequences of these attacks depend upon the environment in which the systems operate. For example, a successful attack against a critical infrastructure may have catastrophic impacts upon human life, environment and economy of a country, to mention a few. Critical systems are favourite of attackers because their malfunction can generate huge loss [4]. According to a report published recently by US Department of Homeland Security, the number of attempted and successful cyber-attacks against Critical Infrastructures (CI), such as dams, energy and water systems rose more than 383% from 2010 to 2011 [5]. In order to mitigate these attackes, the research has been focused on following main aspects:

- finding new techniques to provide the best possible protection
- working on how to produce and retain valid evidence from legal point of view when there is a violation of security rules.

A Security Information and Event Management (SIEM) [6] is a solution that covers both the above research areas by providing the following collection of services: Log Management, IT regulatory compliance, Event correlation, Active response, Endpoint security and Forensic Storage. The presence of forensic storage in a SIEM is very necessary because it helps to retain digital evidence against malicious parties responsible for security breaches. Many free versions of SIEM solutions do not implement any technique to forensically store the security events, although some of them offer this service in their commercial version. Having a forensic storage means to create an infrastructure capable of ensuring the integrity and unforgeability of stored data.

In order to ensure integrity and unforgeability, the forensic storage of the current SIEM solutions use classic RSA[1] algorithm based on a pair of public and private keys to sign the security events. The strength of RSA is based on the problem of factorization of large numbers which is a hard problem and no algorithm exists to solve it in real time. The easiest way to compromise the security provided by RSA algorithm is through the compromise of its secret key. So the easy search of secret key from a hard disk drive [7] used in RSA algorithm can nullify the admissibility of digitally signed messages for forensic purposes. Also if an attacker successfully compromises the signing module, he can stop the signing of security related events.

In this paper, we propose an architecture that overcomes the limitations mentioned above, i.e. it works correctly even when it is under attack and possibly some components are compromised. In order to do so, we have analyzed the state of the art of two SIEM solutions in terms of techniques that they use to store the data forensically. Then, we describe the architecture and the implementation of our forensic storage. The solution we propose relies on the combination of threshold cryptography and replication with diversity to achieve fault- and intrusion tolerance. In order to evaluate the performance cost of the threshold scheme (and of the replication), we have implemented the architecture in an experimental testbed based on a widely used open source SIEM, namely OSSIM [8]. The performance comparison is done in terms of number of signatures generated by OSSIM with our forensic storage and OSSIM with forensic storage based on classic RSA. Results show that the cost can be as little as 5% if system parameters (specifically: number of nodes n and threshold value k) are configured properly. Also through another test, we show the comparison between times taken by Forensic Storage in normal and faulty situations. In particular when Forensic Storage works in faulty situations, it takes about 72% more time than when

[1]Rivest, Shamir, Adleman (RSA) a famous public key crypto-algorithm

it works in normal conditions. Finally using the threshold anomaly criteria that we have introduced, we can reduce this overhead to 7.2%.

II. Background

A. Threshold Cryptography

Threshold Cryptography is not a crypto-system itself but a technique that uses already existing crypto-systems with minor changes. These changes include the distribution of secret key into different shares. It provides more flexibility to the signing process due to the fact that this scheme works correctly also when some secret shares are compromised.

In 1979, Adi Shamir [9] and Blackley [10] proposed independently the first secret sharing scheme. According to this scheme the secret information can be divided in different parts in such a way that all or a certain number of these shares are necessary to construct the secret. Formally, let S be the secret to be split among n participants called shareholders. We want S divided in n shareholders in such a way that $k < n$ shareholders are necessary to construct S but no fewer than k shareholders can. In particular an adversary who gets control of at most $k-1$ shares has zero knowledge of the secret key. Such a scheme is called a (k, n) threshold scheme. The threshold number k is defined such that it is impossible to build the secret if less than k shareholders cooperate together.

In this paper we have used the RSA threshold signature scheme proposed by Shoup [11] because while the previous work is theory oriented, this work is the first practical scheme that provides also implementation details. It is explained mathematically how to generate the secret key shares (SKS) from secret key, the verification key shares (VKS) and complete verification key (VK). So, the SKSs can be used by different shareholders in order to create partial digital signatures of a message. Moreover, the output of a digital signature algorithm in its threshold mode must be equivalent to the digital signature produced when this algorithm is used in a traditional way. Shoup has also explained the mathematical operations necessary to put together the signature shares so to obtain a complete signature equivalent to the standard RSA signature. The VKSs provide a way to check the correctness of each signature share whereas VK is used to check the validity of complete signature.

B. SIEM: State of the Art

SIEM [6] solutions are a combination of the formerly disparate product categories of Security Information Management (SIM) and Security Event Management (SEM). In particular, SEM systems are focused on the aggregation of data into a manageable amount of information with the help of which security incidents can be dealt with immediately, while SIM primarily focuses on the analysis of historical data in order to improve the long term effectiveness and efficiency of information security infrastructures [12].

At the time of writing this paper, there are about 85 existing SIEM solutions [13] some of which are freeware whereas others are commercially available. In this paper our interest is focused on the intrusion and fault tolerant system and integrity and unforgeability of data stored so that it can be used for forensic purposes. So we have analyzed two main SIEMs in order to understand what they offer from the point of view of these requirements. These two SIEMs are Assuria Log Manager (ALM) and OSSIM SIEM system. The forensic storage of ALM is based on digital signatures of the security events with classic RSA [14] with SHA-256 hash function [15]. OSSIM SIEM developed by AlienVault [8] offers a forensic storage component called Logger only in its commercial version. The user manual of OSSIM shows that the Logger uses the classic RSA algorithm to sign the important events.

III. Limits of Current SIEMs and Analysis of Techniques for Forensic Storage

As described in section II-B, most of the current SIEM solutions use traditional RSA to sign the security events. In order to improve the state of the art, we have compared various techniques to sign the security events. Since the purpose is to provide an architecture of forensic storage, our analysis is focused on integrity and unforgeability of data and intrusion and fault tolerant architecture. So we have analyzed three techniques to sign the security events: RSA classic scheme, parallel RSA scheme and Threshold Cryptography scheme. For parallel scheme we mean multiple RSA modules deployed in parallel.

The forensic storage which uses classic RSA algorithm to sign the security events may face a single point of failure problem. In fact, an attacker could perform a DoS attack if he knows the IP address of the signing module, so to stop it from signing the events. So the attacker will be able to bring down the system. Also, an attacker could compromise the node that generates RSA signatures through a malicious software installed on this node and can forge the signatures. In this case, the RSA module works but in a wrong way so the integrity of data stored is not ensured. Also, these data are unacceptable for forensic purposes. The advantage to use the RSA classic module is that it is fast due to the fact that it does not involve huge usage of memory. In fact, when the module receives the input, it provides an output (signature) without maintaining any intermediate information.

A simple variant of traditional RSA can include the usage of multiple RSA modules working in parallel to sign the same message. All RSA modules use different public and secret key pairs. Each module signs the incoming security event with its own private key and sends the signature to an elector element. In order to improve the security level the elector must receive at least k (threshold number) signatures for the same event before verifying the integrity of data. When at least k (quorum) signatures arrive, the elector verifies them through their corresponding public keys and when the quorum of valid signatures is reached the elector chooses one of them and stores it with the data and the public key in the storage media.

The parallel RSA scheme is slower than the classic RSA scheme because the elector, in this case, must maintain an internal state for each security event. The state related to each event is represented by the signatures received that are less than k. In this case the elector also cannot store the event with the selected signature. Another performance penalty compared to the classic RSA scheme is due to the process to verify each received signature. In fact this process is repeated k times for each security event. The advantage of multiple RSA modules deployed in parallel over single RSA module is that the former is tolerant to intrusions and faults. With the attack scenarios described above for single RSA module, this new architecture works correctly. In fact if one RSA module is corrupted by the attacker it will send a wrong signature to the elector. The elector simply verifies and discards this wrong signature and waits for next one. Instead, when module is under a DoS attack, the architecture also works correctly because other modules still send their signatures to the elector.

Finally we have analyzed the Threshold Cryptography scheme. In this scheme the secret key is divided in different shares and given to different parties or shareholders where each shareholder signs the incoming message with its secret share and sends the signature shares to another module called combiner. The combiner element combines all the signature shares and produces a complete signature. Then, the combiner verifies the complete signature and stores in the storage media after successful verification.

When compared to the classic RSA scheme, threshold cryptography has the same advantages and disadvantages of RSA parallel scheme as previously described. In particular, compared to RSA parallel mode, threshold cryptography is faster because in the former scheme the elector must verify at least k signatures of the same message in order to select one of these verified signatures. On the other hand, in threshold cryptography only complete signature must be verified. This is because at first the combiner combines the signature shares and then it verifies the complete signature. The dynamic of combining process is much faster than the verification process, so it can be considered negligible. If a complete signature is not verified, it means that a shareholder has sent invalid signature share. So, the combiner can verify all the signature shares with the help of VKSs in order to identify the shareholders responsible for sending wrong signature shares. In this case the behaviour of threshold cryptography approach is equivalent to the RSA parallel scheme. But in the normal case when there is no any faulty shareholder, the combiner verifies only one complete signature providing better performance. For these motivations, we have used threshold cryptography scheme to build our architecture of forensic storage.

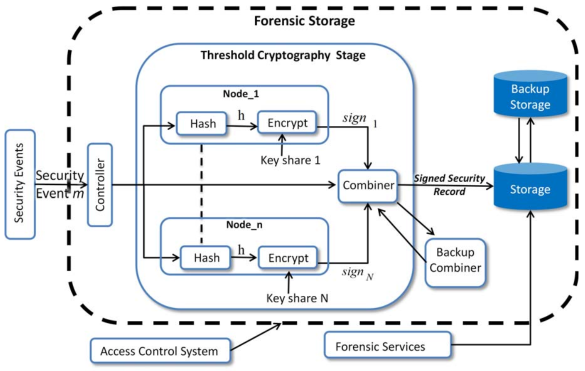


Figure 1. Full architecture of forensic storage

IV. High Level View of Forensic Storage Architecture

In this section we describe the architecture of forensic storage. Since the application domain is SIEM, the forensic storage will contain the critical security events, which require properties like integrity and unforgeability, in order to be used for forensic purposes. The architecture of the forensic storage is shown in Figure 1. The Security Events generated from SIEM system are the input data to the architecture. The incoming data carry information about the events related to a security incident. We assume that the events source is trusted and reliable. After the secret key is divided into shares and distributed to all nodes, the incoming message (containing information about the security breach) is sent to the controller component. The controller component relays the events coming from SIEM to all nodes and to the combiner element. An attacker may compromise the controller and modify the events before they are sent to all nodes. In order to avoid this weakness we use the principles of redundancy and diversity together with voting technique [16] [17]. In fact, we have more than one controller to receive the same event. Then only one event is provided as input to the shareholders after the decision is made through the voting mechanism.

So, each node or shareholder that has received the event computes a hash function of the same message. This function returns a digest for this message, represented by h in Figure 1. The next step is to encrypt the digest with secret key share in order to produce a signature share (or partial signature) and send it to the combiner. The combiner is responsible for assembling all partial signatures of the same message received from the participants of the threshold crypto-system in order to generate a complete signature. After verification, complete signature is attached to the original message, thus forming a signed security record, i.e. a forensic record and is stored for future forensic analysis.

In Figure 1 Forensic Services provide a forensic analysis that is able to determine when the intrusion occurred, which data were altered, and also, who the intruder was, through the information obtained by the recorded events.

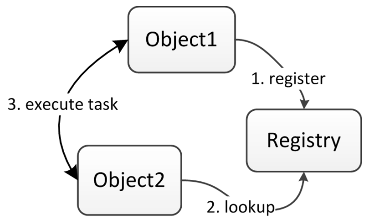


Figure 2. An example of binding and lookup of a service.

The Access Control System is the service that controls who can interact with resources and how. For example, only the system administrator can change the threshold value k or add a new node to the architecture.

In order to improve the intrusion and fault tolerance of forensic storage, replication and diversity are employed. In fact, the combiner and the storage are replicated through a set of software replicas, which are deployed on a set of independent servers. The replication introduces well-known data consistency issues that have been extensively studied but are beyond the scope of this paper.

V. Low Level View of Forensic Storage Architecture

A distributed approach has been used in the design and implementation of forensic storage. The basic idea is to join the advantages of the threshold cryptography with the advantages of classic distributed systems in order to obtain a system that guarantees security of data and tolerance to both intrusions and faults. In fact, the threshold cryptography guarantees the integrity of data if less than k out of n shareholders are corrupted or down for any reason. Also, the distributed system is easily scalable allowing enhancement in the performance.

The design of the forensic storage has been composed in several steps. The first was to identify the main components of the architecture and to export them as a service. In Figure 2 the mechanism of binding is shown between a component that offers a service and a component that is in search of a service. The Object1 aims to offer a service, so it registers its service in the registry. Object2 does not know anything about Object1 but it knows only the name of the service that it needs. Object2 searches for the service into the registry by using the name of the service as a search key. If the specified service exists, registry sends a reference to Object2 to contact service provider, Object1 in this case.. The registry is the only component that maintains a list of all services, so it can be a single point of failure in the architecture. To avoid this weakness, the Reliable Registry of Alberto Montresor is used in the implementation phase [18]. In Figure 3 the main services during initialization phase are shown. More details about the implementation of components are available in [19].

When the combiner is started, it registers its service at the registry and then establishes a connection to the database. When the shareholders are started, they also register their services at the registry.

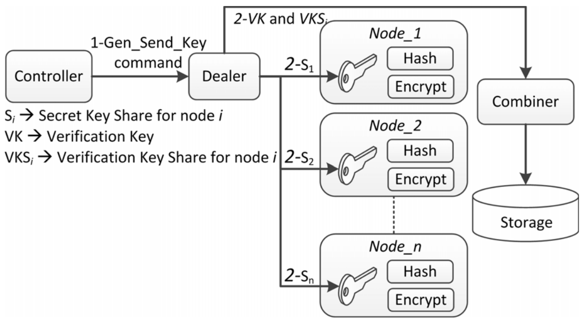


Figure 3. During the initialization phase, the controller sends a request to the dealer to generate and send key shares to all the shareholders and combiner.

In the initialization phase, the controller searches in the registry which component offers the secret key shares generation and distribution service. The registry sends the controller the reference of a remote object that offers the requested service, dealer in this case. Then the controller asks the dealer to generate and send n SKSs to sharesholders and one VK and n VKSs to combiners. Before generating all the keys, the dealer searches for the intended shareholders in the registry that will participate to the threshold cryptography scheme and the components which will provide the service of combining the signature shares. So, the dealer directly distributes each SKS to each shareholder, and it gives the VKSs and the VK to the combiner. When a shareholder receives its key share, it sends the acknowledgment to the dealer. After receiving the acknowledgments, the dealer erases the memory that was used to store the keys. The erasure of the SKSs is very important so that if the dealer is compromised in the future, the adversary will not be able to recover the SKSs from its memory. So we consider to trust the dealer only during the key generation and distribution phase. After this point the initialization phase is complete.

In Figure 4 the normal working of the system when a new security event arrives is shown. The controller distributes the new event to all nodes and combiner. Each node when receives security event, generates a digest value and encrypts this digest values with its SKS creating a signature share. Finally the node sends the signature share to the combiner. When the combiner receives at least k signature shares for each event, it combines them generating a complete signature. Then, the combiner checks if the complete signature is valid though the VK. If complete signature is valid, a new record, containing the original message, complete signature and corrupted nodes (if any) will be created and stored in storage by the combiner. If the complete signature verification fails, the combiner checks each signature share with the correct verification key share in order to recognize which node has sent wrong signature share. The combiner sets a flag true if a signature share is valid otherwise the flag is set to false. So the next time when new signature shares are available for a security event the combiner chooses a set of k signature shares among newly received and signature shares whose flags

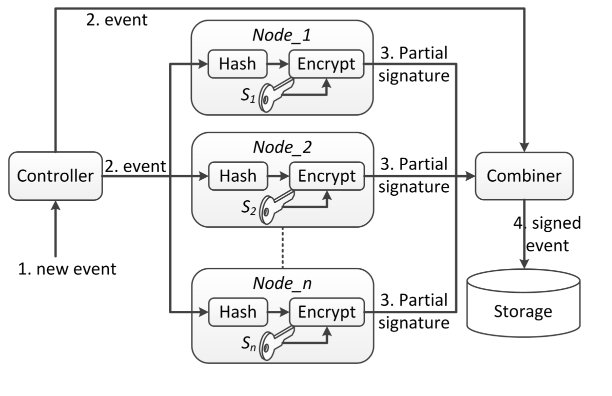


Figure 4. This figure shows how the forensic storage works when it receives a security event.

are set to true. If this time, the process of combining and verification of the complete signature is valid the forensic record is created, otherwise the process is repeated when new signature shares are available.

The technology used to implement the architecture of forensic storage is Java-RMI[2]. In order to improve the level of security when the different services communicate with each other, communication is allowed only through SSL. The authentication mode choice is bilateral. Certificate management in Java is made possible through the use of key-store and trust-store. So all certificates have been pre-generated and stored in the key-store and trust-store. The length of SKS, VKS and VK is 1024 bit. The hash function that each node uses to calculate digest is SHA-256.

VI. Integration of Forensic Storage with OSSIM SIEM and Validation

In order to show the performance of the system and features of intrusion and fault tolerance of forensic storage in presence of attacks, we have integrated it in OSSIM SIEM. Generally SIEMs produce a huge amount of events per second (EPS), so it is not possible to store all events but only those related to security breaches and can be used for forensic purposes. In the SANS study [20] a benchmark is reported measuring the number of events collected by SIEM solution deployed in a mid-sized organization. In this study the authors show that the average EPS generated in normal conditions are 149.79. When two subnets are attacked the EPS value is 8,118.80. So if we consider all the events generated in a large span of time, e.g. 6 months, we have in normal conditions (without any attack) 2.4 billion events. If we use 300 bytes as the average message size, 6 months of data will require about 670 GB of space (if there are no security attacks).

In OSSIM, in order to reduce the number of events, it is possible to correlate and aggregate the events in order to detect real security breaches and attacks. This is possible using the correlation directives [21]. Each correlation directive is composed of correlation rules. All the events are compared with the correlation rules in order to find the pattern of attack. In fact when all correlation rules of one correlation directive are activated, a new event is generated. This event has an associated risk value which can be calculated as:

RISK of the event = (Asset value * Event Priority * Event Reliability) / 25 [22]

where:

- Asset value (0-5). The assets are: Host, Host Groups, Networks and Network Groups
- Event priority (0-5). The Priority is the importance of the event itself
- Event reliability (0-10). Reliability determines the probability of a security event being effectively related to an attack or not.

So only new events generated with a risk value greater than a fixed threshold are significant and must be stored forensically because they are supposed to have information related to attack. When a new event with a risk value greater than threshold is generated, this event contains also the chain of events that previously has matched every single correlation rule belonging to the correlation directive. So, in the forensic storage we store the events related to security breach and the chain of events which has generated them. To integrate the forensic storage in OSSIM we have created a new policy that allows to send the events that have a risk value greater than a certain threshold to a specific server (in our case the controller of the forensic storage). So they will be stored using the architecture described above. OSSIM SIEM offers a forensic storage component called Logger which is available only in the commercial version. Since the Logger is not available in OSSIM free version, we have developed a new module using Java API that signs the events with classic RSA algorithm in order to simulate the behaviour of Logger. We have used the RSA implementation provided by the java.security package that offers the necessary methods to generate a pair of keys and sign the events. Then we have integrated this module in another instance of OSSIM as we have done for our forensic storage. So we have deployed OSSIM with our forensic storage and OSSIM with RSA module to sign the events as shown in Figure 5 which also shows the points where the readings have been taken in order to compare the performance in terms of signature generation time taken by two systems. The setup of the systems used is the following:

- RSA threshold scheme with n=5 and k=3
- hash function used to calculate the digest is SHA-256 both for the nodes of our forensic storage and for classic RSA module
- all keys in our forensic storage and RSA module have a size of 1024 bits.

Each event sent from OSSIM to forensic storage has a size of 1024 bytes and each node has an I3-2330M CPU and 2 GB of RAM. Indeed, the average size of each event is about 300 bytes, but each event contains also the chain of events that has triggered this event. Figure 6 shows the comparison of performance of RSA classic and forensic

[2]Java - Remote Method Invocation. More details are available at http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html

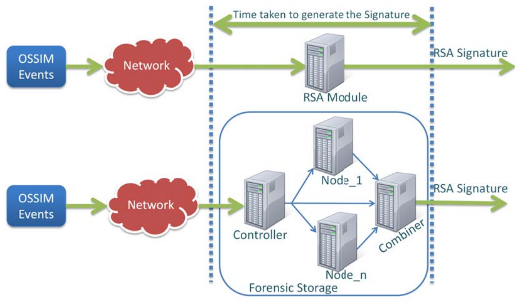


Figure 5. Deployment of OSSIM with two different modules integrated

storage in terms of complete signature generation time. Classic RSA reaches an average signature rate of 510 signatures per second, while forensic storage reaches an average signature rate of 489 signatures per second. So the forensic storage has a penalty in terms of signature rate of about 5%.

This delay is due to the extra time that the combiner takes as compared to the RSA module. The combiner must maintain the signature share of each event until the threshold is reached. When the threshold is reached, the combiner combines the signature shares in order to create a complete signature and after combining process, complete signature must be verified. Instead, in RSA module this overhead is not present. These readings have been taken when there is no any attack. On the other hand, this little overhead that forensic storage takes if compared to RSA classic produces some benefits such as being tolerant to the intrusions and faults.

In order to better explain these features of our system and to validate our architecture, we consider the attack scenario whose time-line is shown in Figure 7(b). We assume the first successful attack occurs after 2 hours of correct functioning of forensic storage. In particular, the attacker compromises a node by using a malicious software hidden in an update of the software pre-installed on the node. So, he can forge the signature share of each security event sent from this node to the combiner. We suppose that the system administrator views logs at the end of each day. The attacker tries to compromise another node with the same technique. Since we have implemented diversity at operating system level and software installed in each machine, the attacker cannot use the same discovered vulnerability. So, we suppose that the attacker takes another 2 hours (time-line now is 4 hours) in order to discover a new vulnerability in another node e.g. he has found a weak password for a user with administrator rights for remote access. We limit our analysis to two attacks because in the setup we have used threshold cryptography scheme with $n = 5$ and $k = 3$, so we can tolerate until $k - 1 = 2$ compromised nodes as described in section II-A.

In our analysis we consider the following scenarios: 1) a compromised node always sends incorrect signature share for each event. Later we explain our strategy of remediation in order to improve the performance when the system

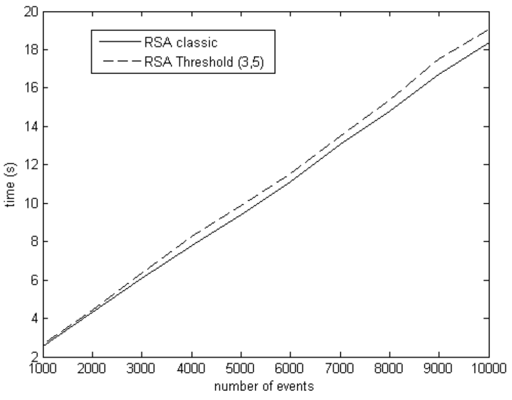
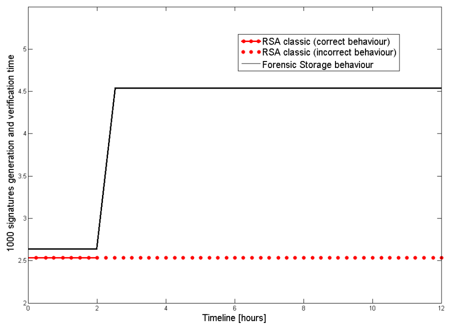
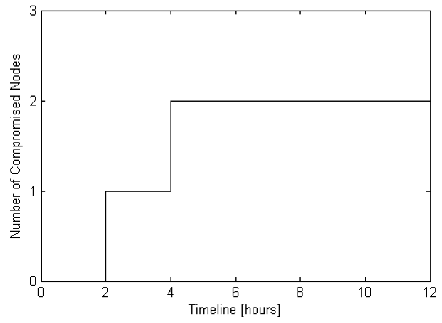


Figure 6. Performance of the forensic storage module and of the RSA standard module

is under attack; 2) the k signature shares chosen for the signing process include all incorrect signature shares. So, the process of verification of complete signature fails and the combiner takes additional time to find a valid complete signature; 3) all the signature shares arrive almost together because they are sent in parallel from each node. The time taken by signature shares to arrive from node to combiner is very small and negligible as compared to the dynamics of the system.



(a)



(b)

Figure 7. An attack model showing the effect of compromised nodes on the performance of forensic storage.

Figure 7(a) shows the overhead in terms of generation and verification time of 1000 complete signatures. In fact using the setup described above in normal conditions (without faulty behaviour) to generate and verify 1000 complete signatures the forensic storage takes 2.639 seconds while RSA takes 2.5350 seconds. In Figure 7(a) we show that the forensic storage, when is under first attack, takes about 4.539 seconds in order to generate and verify 1000 complete signatures. So we take 72% more time as compared to the 2.639 seconds required in normal working conditions. The classic RSA algorithm under the same attack keeps on working with the same signature generation rate but in an incorrect way. In fact it generates bad signatures (dashed red line in Figure 7(a)). Also, when the second attack occurs the forensic storage takes the same time as before to generate and verify 1000 complete signatures. In order to explain the cause of this extra time and why it remains constant when the number of attacks increases, we show in Figure 8 the behaviour of the combiner.

The combiner has two tasks: to combine the signature shares into a complete signature; and to verify the complete signature. In normal conditions (without attacks) the verification process of a complete signature completes successfully and the combiner processes the signature shares for the next event. If the verification process fails, the combiner verifies through the VKSs all k signature shares that have participated to the combining process. Also, the combiner sets a flag to true or false value for each signature share in order to identify if it is valid or invalid respectively. Next time the combiner takes the previous valid signature shares and the new signature shares until it reaches the threshold (k) and repeats the process of combining and verification. So the time that the only combiner takes for generation and verification of C complete signatures is:

$$[(t*(k+1))+(A+B)]*C_{err}+[t+(A+B)]*C_{corr} \quad (1)$$

where:

- C_{err} is the number of events to be signed in faulty situations
- C_{corr} is the number of events to be signed when no faulty signature shares have to be considered
- t is the verification time of each signature
- k is the number of signature shares that must be verified when some nodes are under attack. The term +1 is referred to the verification of complete signature that is always present
- A is the time when a new signature share reaches the combiner. We take this time as negligible
- B is the time necessary in order to combine k signature shares and is negligible for small values of k as in our case
- C is the total number of events to be signed ($C = C_{err} + C_{corr}$).

In our experiment we consider a block of 1000 events to be signed in faulty conditions i.e. when an attacker starts sending two wrong signature shares for each event ($C = C_{err} = 1000$). We can see from formula 1 that the extra time the combiner takes in faulty situations is due to the

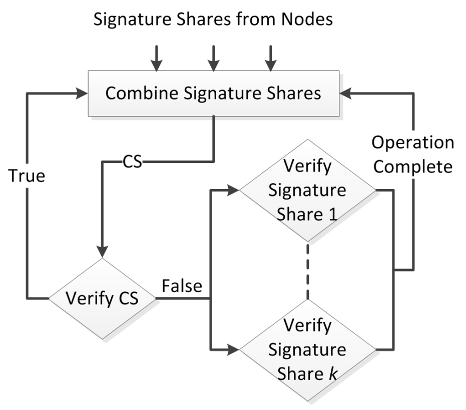


Figure 8. Behaviour of combiner in presence and absence of wrong signature shares

k signature shares that must be verified and the number of events involved C_{err}. So in order to reduce this time in real testbed, we have fixed a threshold of anomalous signature shares for each node. In fact, if a node sends, for example, one wrong signature share and after that it shows a correct behaviour, probably it is not a compromised node. So it is not useful to exclude this node immediately from list of nodes that are considered to be sending valid signature shares. On the other hand if we allow one node to send high number of wrong signature shares the performance penalty is very high. So the choice of the value of the threshold of anomaly is a trade-off between the acceptable performance penalty and the possibility to recognize an incorrect behaviour immediately.

In our case we consider an event window, for example $C = 1000$ and we choose a threshold value of 10%, that is $C_{err} = 100$. In this case, when the threshold value is reached, the combiner stops accepting signature shares from this node for example for the next 4 hours and sends an alert event to the system administrator. Of course the size of events window to consider, the value of the threshold anomalous signature shares, and the time to stop accepting signature shares from a node are set by the system administrator. With this solution the forensic storage under attack takes an extra time of only 7.2% as compared to its own functioning under normal condition. When a second node is compromised, nothing changes in the formula 1. So from one faulty node until $k-1$ the extra time in order to calculate the valid complete signature is constant.

VII. Conclusions

In this work we have shown a comparison in terms of advantages and disadvantages of various techniques in order to use one of them as base of the forensic storage architecture. From our analysis we have found that the best technique that can be used to sign security events is the threshold signature scheme. Also, we have explained the architecture and the implementation of our forensic storage. Then we have integrated the forensic storage in OSSIM SIEM in order to estimate the performance of our system in terms of signatures generation time. Also in order to show the advantages of our architecture in terms of intrusion and fault tolerance we have described an attack model and we have shown that our architecture works correctly. Finally through a test we show the extra time in terms of generation of valid complete signature when the forensic storage is under attack. In order to optimize the performance penalty we have introduced a criteria based on threshold of anomalous signature shares.

The financial cost of our architecture depends upon level of security that must be achieved. In fact, the replication and diversity of components of the architecture have an additional cost that increases with the number of components deployed. Also, if we want to ensure a higher level of integrity we can, for example, add other nodes and raise the threshold k. In each case there will be an additional cost due to new nodes that are deployed and configured. But we must remember that there is a trade-off between security and performance. In fact, in order to ensure the integrity it is possible to add other nodes and to raise the threshold k. So when there are no attacks there is only a greater cost in the combining operation whereas in the presence of compromised nodes, the greater part of the performance penalty is because of verification process of signatures. So the trade-off is among cost, level of security and acceptable performance.

In the future we plan to investigate the possible optimization in order to reduce the performance penalty between our forensic storage and the module based on RSA classic algorithm. Also, we plan to remove the trusted dealer for generation and distribution of keys. Finally, we plan to investigate more attack scenarios i.e. the worst case in which there are z attackers ($z = k - 1$) and each of the wrong signature shares is taken into account in a different set of k signature shares.

Acknowledgments

The research leading to these results has received funding from the European Commission within the context of the Seventh Framework Programme (FP7/2007-2013) under Grant Agreement No. 257644 (MAnagement of Security information and events in Service Infrastructures, MASSIF Project).

References

[1] White Paper, Symantec®Intelligence Quarterly Report: October-December, 2010, "Targeted Attacks on Critical Infrastructures"

[2] White Paper, Global Energy Cyberattacks:"Night Dragon", McAfee®Foundstoner Professional Services and McAfee Labs, (February 10, 2011)

[3] Capasso, V.: "Top Cyber Security Trends, Breaches, and Observations". http://www.myitview.com/security/top-cyber-security-trends-breaches-and-observations Last Accessed 31st August 2012

[4] Symantec®Applied Research. Symantec 2010 Critical Infrastructure Protection Study (Global Results). Oct. 2010.

[5] US Department of Homeland Security, Incident Response Summary Report, http://www.us-cert.gov/control_systems/pdf/ICS-CERT_Incident_Response_Summary_Report_09_11.pdf, June 2012

[6] Carr, D.F.: Security Information and Event Management. Baseline, No. 47, 2005, p. 83.

[7] Shamir, A., Someren, N.: Playing "Hide and Seek" with Stored Keys. Proceedings of the Third International Conference on Financial Cryptography, Springer-Verlag, 1999, 118-124

[8] OSSIM, the Open Source SIEM http://communities.alienvault.com/community/

[9] Shamir, A.: How to share a secret. Commun. ACM, ACM, 1979, 22, 612-613

[10] Blakley, G. R.: Safeguarding cryptographic keys. Managing Requirements Knowledge, International Workshop on, IEEE Computer Society, 1979, 0, 313

[11] Shoup, V.: Practical Threshold Signatures. EUROCRYPT'00, 2000, 207-220

[12] A. Williams, "Security Information and Event Management Technologies", Siliconindia, Vol. 10, No. 1, 2006, pp. 34-35

[13] Log Management & Security Information and Event Management (SIEM), Mosaic Security Research. https://mosaicsecurity.com/categories/85-log-management-security-information-and-event-management Last Accessed 26th April 2012

[14] Rivest, R. L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Commun. ACM, ACM, 1978, 21, 120-126

[15] Assuria Log Manager (ALM), Assuria Ltd. http://www.assuria.com/products-new/assuria-log-manager/features.html Last Accessed 27th April 2012

[16] A. Saidane, V. Nicomette, and Y. Deswarte, "The Design of a Generic Intrusion-Tolerant Architecture for Web Servers" Dependable and Secure Computing, IEEE Transactions on , vol.6, no.1, pp.45-58, Jan.-March 2009 doi: 10.1109/TDSC.2008.1

[17] Li Wang, Zheng Li, Shangping Ren, and K. Kwiaty, "Optimal voting strategy against rational attackers," Risk and Security of Internet and Systems (CRiSIS), 2011 6th International Conference on , vol., no., pp.1-8, 26-28 Sept. 2011 doi: 10.1109/CRiSIS.2011.6061841

[18] Montresor, A.: A Reliable Registry for the Jgroup Distributed Object Model. University of Bologna, 1999

[19] M.Afzaal, C. Di Sarno, L. Coppolino, S. D'Antonio, L. Romano, "A Resilient Architecture for Forensic Storage of Events in Critical Infrastructures", Hase 2012, in press.

[20] J. Michael Butler, "Benchmarking Security Information Event Management" http://www.sans.org/reading_room/analysts_program/eventMgt_Feb09.pdf

[21] OSSIM, Correlation Directives http://www.alienvault.com/wiki/doku.php?id=user_manual:correlation#correlation_directives

[22] AlienVault Risk Metrics http://www.alienvault.com/wiki/doku.php?id=user_manual:dashboards:risk:risk_metrics