Available online at www.koforiduapoly.edu.gh

International Journal of Technology and Management Research 1 (2012) 57-63

International Journal of Technology and Management Research

Server-aided Public Key Signatures for Diverse Network Devices

William Asiedu[1] , K. Osei-Boateng[1], Rajan John[2]

[1]*Dept.of Computer Engineering, Kwame Nkrumah University of Science and Technology, Kumasi.*
E-mail: asiedu2@gmail.com, boat.soe@knust.edu.gh
[2]*Computer Sci. & Engineering Dept, All Nations University College, Koforidua.*
E-mail: rajan.john@allnationuniversity.org

Abstract

One of the main challenges of securing effective computation in diverse network devices tends to be a limitation of their computational power. Server assisted signature scheme was recently presented as nonrepudiation service for mobile and constrained devices. They all tend to have a feature in common: limited computational capabilities and equally limited power (as most operate on batteries). The scheme suffered with high storage requirements and memory requirements for the mobile clients. This makes them ill-suited for public key signatures. This paper examines practical and conceptual implications of using Server-Aided Signatures (SAS) for these devices. SAS is a signature method that relies on partially-trusted servers for generating (normally expensive) public key signatures for regular users. Although the primary goal is to aid small, resource-limited devices in signature generation, SAS also fast certificate revocation, signature causality and with reliable timestamping.

Keywords: Public key infrastructure; Digital signature; Certificate authority.

1. Introduction

Digital signature schemes are among the most fundamental and useful inventions of modern cryptography. In such schemes, each user generates a (private) signing key and a (public) verification key. A user signs a message using his private signing key, and anyone can authenticate the signer and verify the message – using the signer's public verification key. A signature scheme is considered to be secure if signatures on new messages cannot be forged by any attacker who knows the user's public key, but not his private key. Computers and communication networks have become an integral part of many people's daily lives. Systems to facilitate commercial and other transactions have been built on top of large open computer networks. Two crucial features of digital signatures are non-repudiation and strong authentication of both origin and data. While digital signatures are rapidly becoming ubiquitous, one of the major recent trends in computing has been towards so-called smart devices, such as PDAs, cell phones and sensors. Although these devices come in many shapes and sizes and are used for a variety of purposes, they tend to have a feature in common: limited computational capabilities and equally limited power (as most operate on batteries). This makes them ill-suited for complex cryptographic computations such as large number arithmetic present in virtually all public key constructs. In fact, the computation power disparity between the smart devices and the adversary becomes even larger.

At the same time, increased use of digital signatures accentuates the need for effective revocation of cryptographic credentials and certificates, which has been an issue for a long time. However, now the problem is becoming more evident, e.g., the recent Verisign fiasco where a wrong certificate was issued (ostensibly to Microsoft) and its subsequent revocation were both slow and painful. Furthermore, current CRL-based revocation methods scale poorly and are not widely used in practice. Effective revocation is not only useful, but vital in some organizational settings (e.g., government and military) where digital signatures are used on important electronic documents and in accessing critical resources.

Consider a situation where a trusted user (Alice) does something that warrants immediate revocation of her security privileges. Alice might be fired, transferred or her private key has been compromised. Ideally (immediately following revocation) no one

should be able to perform any cryptographic operations involving Alice's private key. In addition, when a cryptographic certificate is revoked (or simply expires), to establish the validity of digital signatures generated prior to revocation (or expiration) becomes a hard issue, due to the difficulty in determining the exact generation time. Albeit a secure time stamping service may provide a means of distinguishing between pre- and post-revocation signature, it hasn't been widely adopted due to its well- known prohibitive cost.

The basic idea of Server-Aided Signatures was introduced as a non-repudiation technique. Its goals are three-fold:

- Assist small, limited-power devices in computing digital signatures
- Provide fast revocation of signing capability
- Limit damage from potential compromise

The signature method (SAS) discussed here is based largely on a weak non-repudiation technique due to Asokan et al. (1999). The most notable feature of the SAS method is the introduction of an online partially trusted entity. Specifically, each SAS signature is generated with the aid of a partially-trusted server called SEM (short for Security Mediator). Informally, the basic SAS signature protocol is as follows:

- First, a prospective signer (Alice) contacts her SEM and provides the data to be signed as well as a one-time token.
- SEM checks Alice's certificate validity and, if not revoked, computes a half-signature over the data as well as other parameters (including the one-time token). SEM then returns the results to Alice. Alice verifies SEM's half-signature and produces her own half-signature. Put together, the two respective half-signatures constitute a regular, full SAS signature. This signature is accompanied by SEM's and Alice's certificates.

The two half-signatures are inter-dependent and each is worthless in and of itself. This is despite the SEM's half-signature being a traditional public key signature: in the context of SAS, a traditional signature computed by a SEM is not, by itself, a SAS signature. The half-signature computed by a user (Alice, in our example) is actually a one-time signature (Shamir and Tauman, 2001) over the other half. Note that computing one-time signature requires little computation resource. Verifying a SAS signature is easy: after obtaining the signature, verifier (Bob) first verifies the correctness of SEM's public key signature, then checks the link between two halves i.e. verifies user's (Alice's) one-time signature.

2. Related works

A well-known technique which is online/offline signatures [1], in this scheme the signing of the message is broken into phases. The first phase is offline. Though it requires a moderate amount of computation; it presents an advantage in that it can be performed leisurely, before the message to be signed is even known. The second phase is online. It starts after the message becomes known, and utilizes the pre-computation of the first phase and is much faster. A general construction which transforms any digital signature scheme to an online/offline signature scheme is presented entailing a small overhead. For each message to be signed, the time required for the offline phase is essentially the same as in the underlying signature scheme; the time required for the online phase is essentially negligible. The time required for the verification is essentially the same as in the underlying signature scheme (Shamir and Tauman, 2001).

In this paper, the signature is based on factoring and DES. In online, DES computation is used because it is ideally suited for electronic smart cards. All the costly computations are performed in the offline stage while the time for the online stage remains essentially unchanged. In some cases the transformed signature scheme is invulnerable to chosen message attack even if underlying digital signature scheme is not. It allows proving that the existence of signature schemes which are unforgeable by known message attack is a sufficient condition for the existence of signature schemes which are unforgeable by chosen message attack. It requires a lot of power for offline computation and this takes a longer time to compute (Asokan *et al.*, 1997).

This provides against denial by one of the entities involved in communication of having participated in all or part of the communication. Actually, non-repudiation prevents either sender or receiver from denying a transmitted message. Thus, when a message is sent, the receiver can prove that the alleged sender in fact sent the message. Similarly, when a message is received, the sender can prove that the alleged receiver in fact received the message.

This is based on one-way hash functions and traditional digital signatures. It is efficient in terms of

computation, communication and storage costs. The existing techniques for non-repudiation are based primarily on either symmetric or asymmetric cryptography. Secure symmetric techniques are computationally more efficient but require unconditional trust in third parties (Mundadugu *et al* 2001). Unconditional means that if such a third party cheats, the victim cannot prove this to the arbiter. Asymmetric techniques are computationally less efficient, but can be constructed in a way that allows one to prove cheating by the third parties involved. Server supported signature for non-repudiation of origin operates in one way hash function. This function operates on arbitrary length inputs to produce a fixed\ length value. A one-way hash function is said to be collision resistance if it is computationally infeasible to fine any two string. Full trusted usually implies poor scalability. Therefore the fully trusted server being a single point of failure in terms of security and availability becomes an attractive target for various attacks in many applications, it is impractical to establish a centralized fully trusted entity. A fully trusted server actually puts the users' security at risk as server compromise exposes all users' secret information.

There have been explosions in the number of applications for handheld devices. Many of these applications come with a remote device over an authenticated channel. Examples of applications include a wireless purchase using a cell phone, remote secure synchronization with a PDA, using a handheld device as an authentication token and handheld electronic wallets. According to authors, generating a 1024bit RSA key on handheld devices like palm-pilot can take as long as 15 minutes (Naor and Nissim, 1998). The device locks up while generating the key and is inaccessible to users. For wireless devices battery life time is a concern. The application may need to generate a key before it can function. Generating the key while the user is traveling will lock up the cell phone for some time and may completely drain the batteries. The obvious solution here is to allow the handheld to communicate with a desktop or server and have the server generate the key. The key can then be downloaded onto the handheld. The problem with this approach is that the server learns the user private key. Consequently, the server must be trusted by the user. Generating an unbalanced RSA key with the help of untrusted servers (Goodrich *et al.,* 2001). At the end of the computation the servers should know nothing about the key they help generate. The assumption is that those two servers cannot exchange information with each other. This ensures that an attacker cannot eavesdrop on the network and obtain the information being sent to both servers. This approach limits mobility of the handheld application since users can only generate a key while communicating with their home domain. Server assisted one time signature scheme was recently presented as a non-repudiation service for mobile and constrained devices. However, the scheme suffered with high storage requirements for the virtual server and high memory requirements for the mobile client. This scheme significantly reduced virtual server storage requirements as well as mobile client memory requirements (Lamport , 1981). More precisely, the virtual server storage requirements in this scheme are reduced by a factor of more than 80 compared to the original scheme. Further, memory requirements for the mobile client are reduced by a factor of more than 130. This is done by generating various quantities pseudo randomly and storing just their cryptographic hash (instead of storing them fully) wherever possible, while still being able to perform dispute resolution. To have some legal significance, these transactions should have some form of non-repudiation (Goyal, 2004). This is usually provided through a digital signature. However, digital signature generation and even verification are known to be computationally intensive processes. It is not always practical to implement public key cryptography and hence digital signatures on a mobile device having limited computational resources and memory. The main problem here is high storage requirements for the verifiable server and memory requirement for the mobile clients. It is easy for the third party to cheat in this setting since it could sign any message on behalf of the user.

3. Background

A. Model and notation

We distinguish among 3 types of entities:

- Regular Users – entities who generate and verify SAS signatures.
- Security Mediators (SEMs) –partially-trusted entities assisting regular users in generating SAS signatures.
- Certification Authorities (CAs) – trusted offline entities that issue certificates and link the identities of regular users with SEMs. SEMs and CAs are verifiable third parties from the users' point of view.

All participants agree on a collision-resistant one-way hash function family H and a digital signature scheme. In SAS, the latter is fixed to be the RSA scheme (Naor and Yung, 1989, Boneh *et al.,* 2001)

SAS description

The SAS system consists of four component algorithms: Setup, Sign, Verify, and Renew. Setup initializes the settings for SEMs and regular users; Sign computes SAS signatures on given messages, which can later be validated by running Verify. Handoff algorithm allows a regular user to switch from one SEM to another (Ding *et al.,* 2001). Renew algorithm allows a user to use new one time private keys (a hash chain as shown below) without applying for a new certificate.

A. Setup

The system administrator sets up a CA and initializes a system-wide cryptographic setting. Specifically, the administrator selects a collision-resistant oneway hash function H( ) for the users. The choices of H( ) include SHA-224 and SHA-256, which are massumed to be secure. A public key signature scheme, which is secure against adaptive chosen message attacks, is selected for SEMs. In order to minimize computation overhead for regular users, the chosen public key signature scheme should be efficient for verifiers. (This is because, as will be seen later, verification is done by regular users, whereas, signing is done by much more powerful SEMs.) Therefore, choose RSA signature scheme (Askan et al 1997,Shamir and Tauman, 2001) with a small public exponent, such as 3 and 65,537 for SEMs.

To become a SAS signer, Ui customizes H( ) into Hui ( ). In essence, Hui ( ) is a keyed hash (e.g., Mackenzie and Reiter, 2001) with a known key set to the identity of the signer. Then, Ui generates a secret random element SK0 i and chooses n as the number of messages to sign. Starting with this value, Ui computation is as follow:

$$\{ SK0^{i}, SK1i, ..., Sk_{i}^{n-1}, SK^{n}_{i} \}$$

where

$$SK^{j}_{i} = H_{ui}(SK^{j-1}_{i}) = H^{j}_{ui}(SK^{0}_{i}) \text{ for } 1 < j < n$$

The hash chain of $\{ SK_{0}^{i}, SK^{1}_{i}, ..., Sk_{i}^{n-1}, SK^{n}_{i} \}$ is called Ui's key chain. Each SKj i , for $0 < j < n$ is Ui's j-th (one-time) private key. It subsequently enables Ui to produce (n-1) SAS signatures, since as shown below, each of them will be used only once. The first value, SK_{0}^{i}, is referred to as U_{i}'s seed private key. The last value, SK^{n}_{i} , together with n, are referred to as $U_{i's}$ root public key Pk_{i}.

Each SEM initializes its own secret/public RSA key-pair (d_{sem}, e_{sem}) of sufficient length. (We use the notation [x]dsem to denote SEM's signature on string x). Each CA also has its own key-pair much like any traditional CA. In addition to its usual role of issuing and revoking certificates a CA also assign associations between users and SEMs by listing SEMs in users' certificates. Each user has a unique Registration SEM in her home domain. Roaming users are allowed to have associations with alternative SEMs in other domains. One SEM serves for a multitude of users. The number and placement of SEMs are expected in an organizational network to closely resemble that of OCSP Validation Agents (Vas) (Kaufmann *et al.,* 1995). In order to obtain a SAS certificate $Cert_{i}$, U_{i} composes a certificate request and submits it to the CA via some (usually off-line) channel. U_{i}'s SAS certificate has, for the most part, the same format as any other public key certificate; it includes values such as the holder's distinguished name, organizational data, expiration/validity dates, serial number, and so forth. Additionally, a SAS certificate contains two other fields:

1. U_{i}'s root public key PK_{i}, i.e $< SK^{n}_{i}, n >$
2. A pair of distinguished name and certificate serial number for each SEM associated with Ui. Once issued, U_{i}'s SAS certificate Certi can be made publicly available via a directory service such as LDAP.

B. SAS signature protocol

To get the first signature from SEM, U_{i} needs to register herself to her assigned SEM either off- or on-line. In the off-line case, SEM obtains U_{i}'s SAS certificate via manual (local or remote) installation by an administrator or by fetching it from the directory service. To register on-line, U_{i} simply includes her SAS certificate as an optional field in the initial SAS signature request to the SEM. Before processing the request as described above, the SEM checks if the same certificate is already stored. If not, it installs in the certificate database and creates a new user entry. To run the same protocol with an alternative SEM, U_{i} must run. In the initial run of the protocol, the signature counter k is set to n-1. Both SEM and Ui

consistently maintain the counter by decrementing it after each run. The protocol is illustrated in Figure 1.

Step 1. U_i starts by sending a request containing: $\{U_i, m, k, SK^k_i\}$ to its assigned SEM. If for privacy reasons U_i does not wish to reveal the message to the SEM, m can be replaced with h(m). U_i may optionally enclose her SAS certificate.

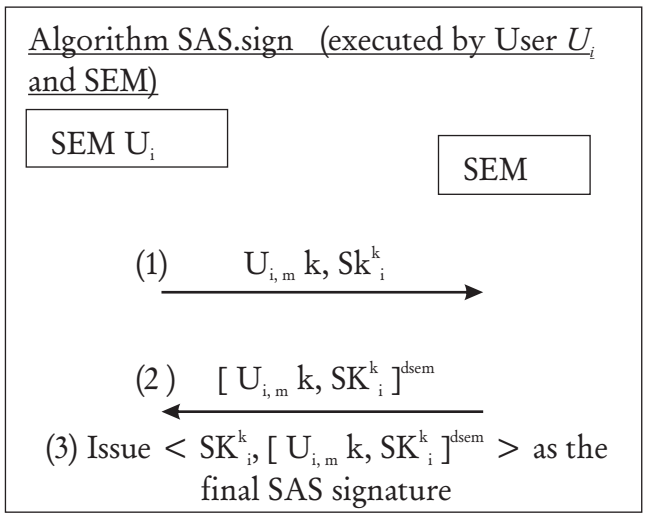

Fig. 1: SAS signature algorithm.

Step 2. On receiving U_i's request, SEM obtains Certi (either from the request or from local storage) and checks its status. If revoked, SEM replies with an error message and halts the protocol. Otherwise, SEM compares the signature counter in the request to its own signature counter. In case of a mismatch, SEM replies to U_i with the half-signature produced in the last protocol run and aborts. (Note that SEM keeps a record of all previously generated half-signatures) Then, SEM proceeds to verify the received k-th "private" key (SK^k_i) with U_i's root public key in Certi. Specifically, SEM checks that Hn-kui (SK^k_i) = PK_i. In case of a mismatch, SEM replies to U_i with the last recorded half-signature and aborts the protocol.

Otherwise, SEM signs the requested message with its RSA private key d_{sem} using RSASSA-PSS scheme specified in Asokan et al. 1997. For simplicity, the result is denoted as $SIG_i = [Cert_i, m, k, SK^k_i]^{dsem}$. Other attributes may also be included in SEM's half-signature, e.g., a timestamp. SEM decrements U_i 's signature counter, records the half-signature and returns the latter to U_i.

In the above, SEM assures that for a given SAS certificate exactly one signature is created for each SK^k_i. This property is referred to to as the SAS Invariant. This concept enables non-repudiation for SAS signatures and protects users from being framed by SEMs.

Step 3. U_i (who is assumed to be in possession of SEM's certificate) verifies SEM's half-signature, records it and decrements her signature counter. If SEM's half-signature fails verification or its attributes are wrong (e.g., it signs a different message than m or includes an incorrect signature counter j ≠ k), U_i aborts the protocol and concludes that a hostile attack has occurred. In the end, U_i's SAS signature on message m has the following format:

$$[Cert_i, m, k, SK^k_i]^{dsem}, SK^{k-1}_i$$

The second part, namely SK^{k-1}_i is U_i's half-signature. As mentioned earlier, it is actually a one-time signature since Hui (SK^{k-1}_i) = SK^k_i. Note that U_i must use her one-time keys strictly in the reverse order of key generation, i.e. starting from SKn-1ii, SK^{n-2}_i, SK^{n-3}_i and so on. In particular, U_i must not request a SEM half-signature using SK^{k-1}_i unless, in the last protocol run, she obtained SEM's half-signature containing Sk^k_i

C. SAS signature verification

SAS signature verification comes in two flavors: light and full. The particular choice depends on the verifier's trust model. If a verifier trusts a SEM to honestly check user requests and verify user certificate status, he can choose light verification. Otherwise, he chooses full verification.

Light verification involves the following steps:

- Obtain and verify $Cert_{sem;}$
- Verify SEM's RSA half-signature: $[Cert_i, m, k, SK^k_i]^{dsem}$
- Verify U_i's half-signature: $H_{ui}(SK^{k-1}_i) = Sk^k_i$

Full verification requires, in addition:

- Verify $Cert_i$ and obtain n from $Cert_i$;
- Check that k < n, otherwise abort;
- Verify U_i root public key: $H^{n-k}_{ui}(SK^{k-1}_i) = Sk^n_i$.

Note that light verification does not involve checking U_i's SAS certificate. Although this may seem counter-intuitive, but SAS signature format (actually SEM's half-signature) already includes $Cert_i$ as assigned attribute. Therefore, for a verifier who trusts the SEM, step 2 above implicitly verifies $Cert_i$. It is

easy to see that, owing to the trusted nature of a SEM and the SAS Invariant, light verification is usually sufficient. However, if a stronger property (such as non-repudiation) is desired, full verification may be used.

D. SAS renewal

A renewal is needed when the messages to sign outnumber the length of the key chain, or the states between SEM and the user is inconsistent due to attacks or system failures. The renewal protocol allows a user to use a new chain of private keys without applying for a new certificate, on the condition that her seed private key is not compromised. Suppose user U_i is currently using the hash chain seeded with SK^0_i and the SEM is expecting SK^k_i . To shift to a new chain seeded with SK^0_i , U_i and SEM run the following protocol:

Step 1: In order to sign w messages in the future, U_i generates a new hash chain of length w + 1: SK^0_i,..... SK^w_i and computes α = Hui (SK^0_i ,SK^w_i). In the protocol message, REN(U_i), a pre-defined macro message indicating U_i's hash-chain renewal request; ω the index of new root public key SK^w_i , SK^k_i is a current hash token to use in the current hash chain; α serves as a commitment to the seed private key of the old chain and the root public key of the new chain.

In this scheme is design to help solve majority of the problem which is faced by smart devices. This scheme provides a solution to the source authentication problem under the assumption that the sender and the receiver are loosely time synchronized. SAS protocol has the following properties; low computation overheads, low communication overheads, if a packet arrives in time the receiver can verify its authenticity. The two half signatures are interdependent, so each is worthless and of itself.

Step 2: SEM checks the authenticity of SK^k_i) and U_i's certificate status as in SAS signature protocol. If the renewal is approved, SEM returns a signature on the request as a normal SAS signature. Meanwhile, the state is updated so that any future SAS signature requests using this chain will be rejected and an attack alarm should be signaled.

Step 3: If SEM's signature in the second round is verified valid, U_i reveals to SEM the SK^w_i and Sk^0_i

Step 4: SEM checks if H_{ui} (SK^0_i ,SK^w_i) equals to α received in the first round. If true, SEM replies with an RSA signature on the SK^0_i and SK^w_i. The signature acts as a special "certificate", which, together with the certificate from CA, are attached with U_i's future SAS signatures.

Table 1
Plain RSA signature timing (ms)

Processor	Key length (bits)			
	1024	2048	4096	8192
PI-233MHz	40.3	252.7	1741.7	12490.0
PIII-500MHz	14.6	85.6	562.8	3873.3
PIII 700MHz	9.2	55.7	377.8	2617.5
PIV-1.2MHz	9.3	58.7	401.2	2835.0

Table 2
SAS Signature Timing (ms)

Processor	Key length (bits)			
	1024	2048	4096	8192
PI-233MHz	13.3	52.4	322.5	2143.4
PIII-500MHz	9.1	46.3	302.0	2070.2
PIII 700MHz	8.5	45.1	377.8	2059.6
PIV-1.2MHz	8.5	45.4	401.2	2061.0

4. Conclusion

The above protocol is implemented in java and it is run on various Speed of Pentium processors. All the experiment was conducted over 100Mbps Ethernet LAN in a Lab, run a number of tests with various hardware platforms and different RSA key sizes and also implement this protocol in Email system, where mail will be encrypted and sent. When ready to send, the user's SAS certificate and extracts the SEM address. SAS-signed emails can be verified by any S/MIME capable email client such as Netscape or Microsoft. The result obtained after running the SAS protocol and RSA on different Systems have been presented.

References

Asokan, N. , Tsudik, G., and Waidner, M. (1997). Server-Supported Signatures. *Journal of Computer Security*, Vol. 5, No. 1.

Boneh,D., Ding,X., and Tsudik,G. ,(2001). Identity-Based Mediated RSA. Advances in Cryptology –CRYPTO ' 2001.

Ding, X. , Mazzocchi ,D., and Tsudik, G., (2002). Experimenting with Server-Aided Signatures, in *Proceedings of NDSS 2002*.

Even ,S., Goldreich ,O., and Micali, S., (1996) . Online/offline Digital Signatures. *Journal of Cryptology*, Vol. 9,No. 1, pp. 35 –67.

Goodrich, M. , Tamassia,R., and Schwerin, A., (2001). Implementation of an Authenticated Dictionary with Skip Lists and Commutative Hashing, in *Proceedings of DARPA DISCEX II*.

Goyal, V. (2004). More Efficient Server Assisted One Time Signatures. Available at http://eprint.iacr.org/2004/135

Kaufman,C., Perlman, R. , speciner ,M., (1995). *Network Security Private Communication in a Public World*. Prentice Hall series in Networking and Distributed Systems.

Lamport, L. , (1981). Password Authentication with Insecure Communication. *Communications of the ACM*, Vol. 24, pp. 770-772.

MacKenzie ,P. ,and Reiter,M. K., (2001). Networked Cryptographic Devices Resilient to Capture, in *Proceedings of the 2001 IEEE Symposium on Security and Privacy*, pp. 12 -25.

Micali, S. , (1996). Enhanced Certificate Revocation System, *Tech. Rep. TM-542b, MIT/LCS6*.

Modadugu, N. , Boneh, D. ,and Kim, M., (2001). Generating RSA keys on a Handheld Use an Untrusted Server, in RSA *Conference, Cryptography Track 2001*.

Naor, M. ,and Nissim, K. ,(1998). Certificate Evocation and Certificate Update, in Proceedings 7th USENIX Security Symposium (San Antonio, Texas).

Naor, M. , and, Yung, M. , (1989). Universal one-way Hash functions and their Cryptographic Applications. *Proc.21 ACM Symp*. On Theory of computing , pp. 33-43.

Shamir, A. , and Tauman, Y., (2001). Improved Online/offline Signature Schemes, in Advances in *Cryptology - CRYPTO '2001*, pp. 355-367.