Available online at www.sciencedirect.com
SCIENCE DIRECT®
Computer Networks 47 (2005) 351–366
COMPUTER NETWORKS
www.elsevier.com/locate/comnet

Improved server assisted signatures

Kemal Bicakci [a,*,1], Nazife Baykal [b]

[a] Department of Computer Science, Vrije Universiteit, De Boelelaan 1083, 1081 HV Amsterdam, Netherlands
[b] Informatics Institute, Middle East Technical University, 06531 Ankara, Turkey

Received 6 April 2004; received in revised form 9 August 2004; accepted 9 August 2004
Available online 22 September 2004

Responsible Editor: D. Frincke

Abstract

It is well known that excessive computational demands of public key cryptography have made its use limited especially when constrained devices are of concern. To reduce the costs of generating public key signatures one viable method is to employ a third party; the server. In open networks, getting help from a verifiable-server has an advantage over proxy-based solutions since as opposed to proxy-server, verifiable-server's cheating can be proven.

Verifiable-server assisted signatures were proposed in the past but they could not totally eliminate public key operations for the signer. In this paper, we propose a new alternative called SAOTS (server assisted one-time signatures) where just like proxy signatures generating a public key signature is possible without performing any public key operations at all. This feature results in both computational efficiency and implementation simplicity (e.g. a reduction in the code size) of the proposed protocol. In addition, SAOTS is a more promising approach since the signature is indistinguishable from a standard signature, no storage is necessary for the signer to prove the server's cheating and the protocol works in less number of rounds (two instead of three). On the other hand, the drawback of SAOTS is the increased bandwidth requirement between the sender and server.

Keywords: Server-assisted signature; One-time signature; Digital signature; Nonrepudiation; Network security

1. Introduction

Rapidly becoming ubiquitous in many spheres of computing, digital signatures are among the most fundamental and valuable building blocks of modern cryptography. There are three security services a digital signature provides:

* Corresponding author. Tel.: +31 20 444 7837/7956; fax: +31 20 444 7653.
E-mail addresses: kemal@few.vu.nl (K. Bicakci), baykal@ii.metu.edu.tr (N. Baykal).
[1] The work was performed when he was in Middle East Technical University, Turkey.

1389-1286/$ - see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.comnet.2004.08.008

- *Authentication*—assurance of the identity of the signer.
- *Data integrity*—assurance that the data is not altered after it is signed.
- *Nonrepudiation*—blocking a sender's false denial that he or she signed a particular data, thus enabling the receiver to easily prove that the sender actually did sign the data.

While there are other means like message authentication codes (MACs) to ensure data integrity and authentication, digital signatures are better in one important respect. They can be used to solve the nonrepudiation problem. Moreover, the MAC approach is inadequate in a multicast setting because it is based on a shared secret among participants.

In recent years, one other major trend in computing has been towards an environment where computer applications will be hosted on a wide range of platforms, including many that are small, mobile and regarded today as devices having only limited computational capabilities. This *pervasive computing* vision could bring a great deal of convenience but also great deal of security risks. For instance think about the possible consequences when the integrity of the data collected from a remote health monitoring sensor is not protected.

Most current techniques for generating digital signatures are based on public key cryptography (based on complex mathematical problems such as factoring or discrete logarithms e.g., RSA [1] or DSS [2]). These traditional methods are simply untenable from a performance perspective when constrained devices are of concern. For instance on a Pentium I—200 MHz machine, [2] using 1024-bit keys, RSA takes around 59 ms to sign and 14 ms to verify. Moreover, some mobile devices may have 8-bit or 16-bit microcontrollers running at very low CPU speeds, so public key cryptography at any kind may not be even an option for them. [3]

One way to reduce the computation cost on mobile/constrained devices is to employ a verifiable and a powerful server. Getting help from a verifiable-server has an advantage over proxy-based solutions (using a fully trusted server) in open networks since as opposed to proxy-server, verifiable-server's cheating can be proven.

Asokan et al. [5] proposed an efficient verifiable server assisted signature protocol called SAS but it does not totally eliminate public key operations for the signer (the signer does not need to generate but instead verify a public key signature). In this paper, we propose a new alternative called SAOTS (server assisted one-time signatures) protocol [4] where just like proxy signatures generating a public key signature is possible without performing any public key operations at all. Eliminating public key operations has a number of attractive benefits including

- *Computational efficiency*: We have already seen that public key signatures are computationally infeasible to generate. In terms of delay for key generation, public key cryptography is even more problematic. For instance, generating a 1024-bit RSA key on a Palm with a DragonBall processor takes 15 min on average [4].
- *Implementation simplicity*: Because algorithms are too inefficient to sign long messages, to save time, digital signatures are usually implemented with hash functions, which convert the long message into a fixed length smaller output. This hash value not the message itself is signed and verified. So traditionally to implement a signature algorithm one needs to implement a hash function and a public key signature algorithm. SAOTS would enable the user to generate a public key signature by implementing only a hash function. This feature would lead a much simpler implementation i.e. small code size.

[2] Today's high-end PDA's and palmtops have a processor speed of 200 MHz.

[3] For instance 1024-bit RSA signature generation takes around 17.45 s using Motorola's 16-bit processor MC68328 DragonBall [3]. We have calculated this performance figure by multiplying the given 'Time consumption for 128-bit multiply result' and 'Number of 128-bit operations required for signing'.

[4] Actually, we will show that SAOTS protocol has three variants: transparent SAOTS [6], SAOTS with server's statement [7], and SAOTS with hash chains [8].

- *Advantage for certain export restrictions*: Because of the fact that the signature algorithm cannot be used as an encryption algorithm.

Besides these benefits, while we are designing our new protocol, we take into account the storage constraints of pervasive devices as well i.e. no signature storing is required for the signer to prove the server's cheating.

Additionally, in SAOTS since the signature is indistinguishable from a standard signature, receivers can transparently verify the signature. The last but not the least, executing in less number of rounds (two instead of three), SAOTS is a more round-efficient protocol than SAS. On the other hand, the drawback of SAOTS is the increased bandwidth requirement between the sender and server.

The rest of this paper is organized as follows. In the next section, related work on efficient digital signature constructions is given. We propose our new signature protocol called SAOTS in Section 3. Section 4 is reserved for the security analysis of the proposed protocol. In Section 5, we give the results of our performance evaluation study. In Section 6 we introduce a variant of SAOTS where the length of messages exchanged are shortened. In Section 7, we discuss the other important issue in server assisted signatures, the issue of revocation of public key certificates. In Section 8 we end by summing up our work and discussing future possibilities for research.

2. Related work

2.1. Efficient public key signatures without server

One possible way to construct efficient signatures would be to relax the security requirements. If a certain amount of risk is acceptable, then one could use less-studied signature algorithms. This method might provide efficiency but if the underlying cryptographic assumptions later turn out to be invalid, these signatures become completely open to compromise.

A more secure solution to speed up the operation of public key signatures is to do the most of the computations on background as precomputations. The second (on-line) phase is performed once the message to be signed is known and is supposed to be very fast so that the response time of signing using a mobile device would be in acceptable range. While some signature schemes can be naturally partitioned into these two phases e.g. DSS [2], on-line/off-line signature schemes were introduced by Even et al. [9] to convert any signature scheme into the aforementioned two phases.

While in the original proposal [9], the length of signatures are longer since one-time signatures (will be explained in the next subsection) are used, the authors in [10] introduce an alternative scheme where the signature size does not increase that much with a trade of heavier on-line computation requirement.

Due to constraints of devices used in signature generation, we might want to minimize or eliminate the amount of public key operations no matter it is off-line or not. If this is the case on-line/off-line signatures are not the solution we are looking for.

2.2. One-time signatures

One-time signatures (OTS) provide an attractive alternative to public key based signatures. Unlike signatures based on public key cryptography, OTS is based on nothing more than a one-way function (OWF). Examples of conjectured OWFs are SHS [11] and MD5 [12]. [5] OTSs are computationally more efficient since no complex arithmetic is involved. Additionally, since OTS is based only on OWF whereas public-key signatures are based on complex mathematical problem as well as OWF, using OTSs allows us to eliminate one more point of vulnerability.

The OTS concept is very easy to grasp [13]. Broadly speaking, a message sender prepares for a digital signature by generating a random number r, which is retained as the private value. He then securely distributes the hash of r, $h(r)$, where h is a one-way function; this represents the public

[5] SHS and MD5 were originally designed as hash functions but they can easily be used as one-way functions when the input message length is set to be equal to the length of output.

value and is used by receivers as the signature certificate to verify the signature. The signature is sent by distributing the value r itself. Receivers verify that this message could only be sent by the sender by applying h to r to get $h(r)$. If this matches the value for $h(r)$ in the signature certificate, then the OTS is considered to be verified, since only the sender can know r. This, in effect, allows the signing of a predictable 1-bit value. In order to sign any 1-bit value, two random numbers (r_1, r_2) are needed; this way, both $h(r_1)$ and $h(r_2)$ are predistributed but at most one of (r_1, r_2) is revealed as a signature. In the original proposal [13], 80 random numbers out of 160 are revealed as the signature of 80-bit hash value of any given message.

Despite the performance advantages provided by OTSs, they have not gain much attention in security world. Other than nontechnical reasons, we believe two disadvantages of OTSs were in effect.

First of all, one-time signatures since they are longer than traditional signatures results in more serious storage and bandwidth constraints. Recent studies succeeded in decreasing the length of one-time signatures in some extent. The authors in [14] realized that p out of n random numbers are sufficient to sign a b-bit length message if the following inequality holds for a given n and p. This is because for any n, we can obtain a valid message mapping by drawing from all subsets containing $p < n$ random numbers:

$$2^b \leqslant C(n,p) = \frac{n!}{p! \times (n-p)!}. \quad (1)$$

To sign an arbitrary length message by OTS, just like the public-key based signatures, we can reduce the length of the message m by computing the hash value of the message, $h(m)$ and then sign $h(m)$. This means for instance for $b = 80$, n must be at least 84 with subsets of size 39 ($p = 39$).

We think the second disadvantage of one-time signatures is a more serious one; they can be used to sign only one message per one public key in its simple form. Since the public key requires to be distributed in a secure fashion which is done most typically using a public key signature, the benefit of using quick and efficient hash function is apparently lost.

There is also a bunch of clever approaches to overcome this limitation. One of which we have already mentioned is on-line/off-line signatures where the public key of one-time signatures is signed by using public key techniques off-line before the message is known. When the message to be signed is in hand, there will not be any necessity to perform public key operation so that the response time (real-time efficiency) is improved.

Due to constraints of the mobile device, we might want to minimize the number of public key operations no matter it is off-line or not. Then Merkle's proposal [15] can be preferred where one-time signatures can be embedded in a tree structure, allowing the cost of a single public key signature to be amortized over a multitude of OTS. The problem in this formulation is the longer lengths of signatures. Now we face a more severe storage and bandwidth requirement than one-time signatures in its simple form since the length of signatures increases as the number of signatures generated using the tree structure increase.

2.3. Signatures employing a server

The third and the last approach to use one-time signatures more than once by using a single predistributed public key is the SAOTS protocol we propose in this paper which is based on a third party (server). But before introducing it, in a more general view we would like to summarize the work on employing a powerful server to decrease the computation requirements for a digital signature.

Server assisted signatures can be explained in three subgroups with respect to the trust relationship between the user and the server. More specifically the server employed may be

- fully trusted (proxy),
- untrusted,
- verifiable.

In the first category, after receiving an authenticated message from a user (a MAC algorithm which can be implemented very efficiently may be used for authentication), a more powerful proxy server on behalf of the user generates a public key digital signature for the message [16]. Notice

that the user himself does not need to perform any public key operation, he just computes a MAC using secret key cryptography. The drawback here is that this simple design is only applicable when the user fully trusts the proxy server i.e. the server can generate forged signatures and that cheating cannot be proven.

In the literature, this method is sometimes called 'proxy protocol with full delegation'. Proxy signatures have other variations mostly designed for the purpose of restricting the server's signing rights. These variations are less-efficient than traditional public key signatures hence are not of interest in our discussion [17].

As the opposite to proxy servers, a totally untrusted server might be utilized i.e. the server only executes computations for the user. Now the goal of securely reducing computational costs on the sender's machine becomes more difficult to accomplish and in fact most of the schemes proposed so far have been found not to be secure. For instance the protocol proposed by Bequin and Quisquater [18] was later broken by Nguyen and Stern [19]. Up to our knowledge, for RSA signatures, designing a secure server-assisted protocol that utilizes an untrusted server is still an open problem. But the situation for DSA is not the same. A secure (unbroken) example for DSA is the interesting approach of Jakobson and Wetzel [20]. However, we see that in their approach public key operations although in reduced amount are still needed to be performed on the constrained device.

2.4. Verifiable-server assisted signatures

The last alternative is to employ a verifiable server (VS). A VS is the one whose cheating can be proven. This approach can be considered in somewhere between the other two since the server in this case can cheat but subsequently the user would have the ability to prove this situation to other parties (e.g. an arbiter). We see that in some papers, the verifiable server is also named as semi-trusted server.

In traditional methods of digital signature generation, the signer usually obtains a public key certificate from a certification authority (CA). In order to trust the legitimacy of signatures, the receiver must trust the CA's certificate-issuance procedures. For instance, the CA can issue a fake certificate for a particular user and then impersonate the user by generating a forged signature. However, if some kind of contract was signed in the certification process, in dispute the signer can prove the CA's cheating by asking this contract from the CA. Notice the similarities between the trust relationship between the signer and CA in traditional methods and the signer and the server in verifiable-server assisted signature protocols.

The first work that aims to reduce the computational costs to generate digital signatures for low-end devices by employing a powerful VS is SAS protocol [5]. In [21], the authors extend this work by providing implementation results as well as other details of the scheme. The scheme in [22] also utilizes a semi-trusted server to generate signatures but their goal is not to minimize the computation cost on low-end machines but to provide fast revocation without losing transparency for those who verify signatures. This work also has the advantage of supporting revocation not just for signatures but for (public-key) encryption as well. We will explore the issue of revocation in Section 7.

We now would like to provide a brief summary of SAS protocol (for a comprehensive treatment, refer to the original papers [5,21]).

There is an initialization phase in SAS where each user gets a certificate from an offline certification authority for K^n (the last element of a *hash chain* of length n) where

$$K^n = h^n(s) = h(K^{n-1}). \tag{2}$$

In Eq. (2), $h(\cdot)$ is a one-way function like SHS [11] and $h^n(s)$ means we apply hash function $h(\cdot)$ n times to an initial input s to generate a *hash chain* of length n. In addition, each user should register to a VS (which has the traditional public-key based signing capability) before operation. Then the SAS protocol works in three rounds as illustrated in Fig. 1:

1. The originator (O) sends m and K^i to VS where
   - m is the message,
   - K^i is the ith element of the hash chain. The counter i is initially set to $n-1$ and decremented after each run.

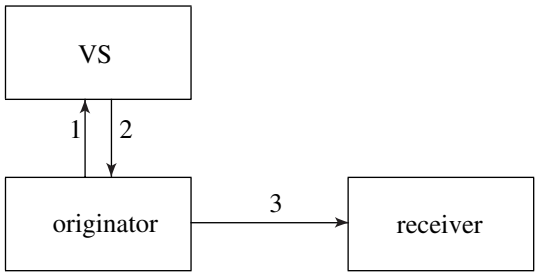

Fig. 1. Asokan et al.'s SAS protocol operating in three rounds.

2. Having received O's request, VS checks the followings:
   - Whether O's certificate is revoked or not.
   - Whether $h^{n-i}(K^i) = K^n$ or in a more efficient way $h(K^i) = K^{i+1}$ since K^{i+1} has already been received.

   If these checks are OK, VS signs m concatenated with K^i and sends it back to O.
3. After receiving the signed message from VS, O verifies the VS's signature, attaches K^{i-1} to this message and sends it to the receiver R.

Upon receipt of the signed message, the receiver verifies VS's signature and checks whether $h(K^{i-1}) = K^i$.

2.5. SAS protocol weaknesses

We have observed that SAS protocol has several drawbacks. These are

1. *Verifying VS's signature*: In step 3 of the SAS protocol, before sending the signed message to R, O should verify the VS's signature otherwise an attack can be performed as follows:

   An attacker modifies the message that O sends to VS and if VS signs this new message instead, O's revealing of K^{i-1} without verifying VS's signature results in a forged signature for the message the attacker has generated.

   Remember that for some pervasive devices restricted with CPU or other constraints, public key cryptography is simply untenable no matter it is used for signing or verifying.

   For more powerful devices if the VS uses RSA [1] signature scheme, where verification is much more efficient compared to signing, this might be affordable. However there are other popular digital signature schemes like DSS [2] where verification is at least as costly as signing so a protocol which offers lightweight signing without any restriction in the digital signature scheme used would be much more flexible and attractive.
2. *Incompatible verification*: As stated in [21], unlike the proxy signatures explained in Section 2.3, SAS signatures are not compatible with other primary signature types. Therefore, the receiver must utilize the custom-built verification method of SAS protocol.
3. *Network overhead*: One of the delay factors of the SAS protocol is the round-trip delay between O and VS. To decrease the network delay, one can try to decrease the number of rounds in SAS, however if O attaches the hash element K^{i-1} to the first message he has sent to VS, an attacker can forge a signed message easily by modifying the message while in transit.

   As a result SAS protocol cannot be a two-round protocol like the SAOTS protocol that will be introduced in the next section this is basically because the signature is not binded with the message itself in a two-round case.
4. *Storing VS's signatures*: In SAS protocol, the signer is required to store VS's signatures to prove its cheating [5]. For some pervasive devices which has a limited storage capacity, this also might put a burden on the operation.

3. The proposed SAOTS protocol

In this section, we propose the server assisted one-time signature protocol (SAOTS) which operates in two rounds as opposed to three. SAOTS is the first VS based approach where the user does not need to perform any public key operation at all. SAOTS might be completely transparent to verifiers (the signatures are indistinguishable from standard signatures). Moreover, in our proposed protocol unlike other alternatives the server not the user is required to save the signatures for dispute resolution. Thus SAOTS eliminates all the four aforementioned drawbacks of SAS protocol.

3.1. The basic idea

Our protocol is built on top of one-time signature idea. Similar to proxy signatures where an efficient MAC algorithm is employed to establish an authentic channel between the user and the server, in SAOTS the user sends the message to the server after he signs it with a one-time signature. The protocol uses the reliable delivery service provided by the transport layer underneath.

In another view, the server serves as the signature translator where it translates the one-time signature of the user to a standard public key signature. However this basic idea of *signature translation* needs to be enhanced otherwise we face again with the inherent problem of OTS, signing only one message per one public key. We will explain how we have solved this problem in subsequent paragraphs. One other solution for this problem that gets benefit of the idea of hash chains is summarized in Section 6.

3.2. Setup

As a setup, each user generates a one-time private key (random numbers) and a one-time public key (hash value of these random numbers) and in a secure fashion he distributes the public key to the server. This can be accomplished by a public key signature if he has already a capability of traditional signing or he can directly get a certificate from a certification authority (CA) for the one-time public key he has. Similarly, the server obtains a certificate from a CA for its public key.

In addition, just like the SAS protocol, each user should register to a VS before operation. In this registration, if 'transparent SAOTS' (will be explained in the next subsection) is the preferred mode of operation then the server generates a private key on behalf of each registered user and obtains a certificate from a CA for the corresponding public key (in the certification process the user confirms that the public key belongs to himself).

3.3. Operation

The protocol works in two rounds as illustrated in Fig. 2:

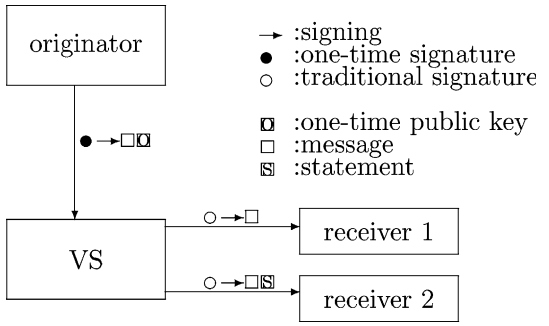

Fig. 2. Operation of the server assisted one-time signature (SAOTS) protocol.

1. The user precomputes a second one-time private key–public key pair. When the message to be signed is ready, he concatenates the message with the new public key and signs this by his previous one-time private key. He then sends the message and the new public key as well as the one-time signature to the server.
2. Having already received securely the one-time public key of the user's signature on the message, the server verifies the one-time signature. He stores the new public key the user has signed for the verification of next message. It will then sign the message with traditional public key techniques. There are two modes of operation for this signing:
   (a) *Transparent SAOTS*: If the user's certificate is not revoked, the server signs the message with the user's private key. Since the signature is indistinguishable from a standard signature, receivers can transparently verify the signature by using the user's public key.
   (b) *SAOTS with Server's Statement*: In some applications the server's public key can be initially embedded in the verification software and the receiver himself can not obtain securely the public key of all possible signers. Then it is better to have the server to sign the message with his own public key (if the sender's certificate is not revoked) after appending a statement on the message saying that it has received it from the user. Another advantage of this method is that the server avoids to obtain a new certificate for each registered user.

One can easily prove that this protocol provides all the three security services asked from a digital signature but only if the server does not cheat i.e. it does not sign any message on behalf of the user without user's approval. We will show in the next section how the server can prove that it is not cheating. If it can prove, the other parties conclude that the user is the one who actually sends the message.

The user can sign any further messages easily by repeating the step 1. The server can always verify the one-time signature since it has securely received the public key in the previous run of the protocol. The server should store all the previous messages for the secure operation but the user does not need to store anything to prove the server's cheating. [6] This becomes clearer after the security analysis given in the next section.

We would like to point that the *chaining technique* we use that attaches the public key for the next message to the current message is first suggested by Gennaro and Rohatgi [23] for signing infinite length digital streams.

4. Security analysis

In this subsection, we show

1. Underlying components (signature algorithms) of SAOTS are secure.
2. How a dispute can be resolved in SAOTS protocol.

At the end we say a few words on another issue in security; the strength of SAOTS to the denial of service attacks.

4.1. Security of underlying components

For secure operation, we need to prove the security of signatures of both the sender and the server. Since the server's signature is a traditional one, we conclude that if the traditional signature algorithm used is a secure one, then the server's signature is also secure.

Secondly, we note that the security of the chaining technique used in the sender's signature has been studied previously. For the security proofs we defer interested readers to [23].

4.2. Dispute resolution

Provided that the underlying signatures building the protocol are secure, we now want to show how a dispute can be resolved. In case of a dispute, the receiver can submit the message and its signature received from the server to an arbiter. The arbiter will verify the followings:

- the signature is valid,
- the public key is certified by the CA,
- the message contains a statement saying that it is originated from the claimed sender (if 'SAOTS with server's statement' is used).

If these checks are successful, then the sender is allowed to take the opportunity to repudiate the message. There will be two checks to decide whether the sender's claim is true or not:

- CA will be asked to prove that the sender's one-time public key was registered by himself.
- The server will be asked to prove that the message was signed by the sender himself.

As a proof, the server shows all the signed messages received from the sender starts from the first one and continues until the message in question is reached. The arbiter verifies all these one-time signatures.

If both CA and server successfully shows that they did not cheat, the arbiter concludes that the sender is dishonest and claims falsely that he has not sent the message.

4.3. Denial of service attacks

In previous server assisted signature protocols, unlike traditional signature schemes, denial of service (DoS) attacks aiming to deny the server's

[6] The server is liable if it has lost some of the signatures. This liability is again similar to CA's liability where CA should not lose certification contracts.

service to the users are of concern. The basic idea behind these attacks is as follows:

By sending legitimate (well-formed) requests, an adversary can force the server to perform alot of signing tasks so that it cannot response timely to the real request coming from users [21].

However if our proposal is preferred, an adversary can not force the server to perform signing. The server can detect illegitimate requests by checking the one-time signature (performing at most p hash computations).

In SAOTS, an adversary can not deny the server's service but the server itself might choose not to give service to a sender. Moreover unlike SAS protocol, in SAOTS the sender can not be aware of the fact that the message was not signed because the protocol is running in two-rounds. Fortunately, we can avoid this cheating by a simple trick. The trick works as follows: The sender sends his message not only to the server but also to the receiver (multicasting might be used). If the receiver does not get the signature of the message from the server, it notifies the sender so that the sender can realize that there is something wrong with the server.

5. Performance evaluation

5.1. Computation and communication comparisons of SAS and SAOTS protocols

Rule #5: You can buy more bandwidth but not lower delays [27, p. 564].

Table 1 shows the comparison of SAOTS and SAS protocols with respect to on-line computational requirements on the participating entities (off-line precomputations are not included).

Note that, in [14], the authors presented an efficient method which costs less than one hash operation for encoding a message for one-time signature. Encoding a message does refer to computation of which subset of random numbers (p out of n) should be revealed as the OTS of the message.

In SAOTS protocol, the server needs to perform one hash to get the hash of the message and p hash operations to verify the OTS if all of n hash values constitute the public key. By a simple trick and with a cost of additional hash operation for the server we can reduce the length of public key to a single hash value (thus the server performs $p + 2$ hash computations in total). The idea is simple: as the public key, calculate the hash of concatenation of all the n hashes. Now to be able to verify the OTS the sender should send the chosen p random numbers and the other $n - p$ random number's hash value. In each run of the protocol the user should send one signature and one public key so if the length of random number is equal to the length of hash value, in overall the signer should send a block of a length of $n + 1$ hashes to the server.

Table 1
Computational comparison of SAS and SAOTS protocols

	SAS	SAOTS
Originator	$1H + 1V$	$1H + 1M$
Server	$2H + 1S$	$(p + 2)H + 1M + 1S$
Receiver	$1V + 2H$	$1V + 1H$

H: hash computation, S: traditional signing by a public key, V: verification of public key signature, M: mapping computation (costs less than one hash), p: number of hash computations to verify OTS.

Table 2 again makes a comparison between the two protocols but now in terms of communication efficiency. As seen from this table, SAOTS provides more efficiency with respect to number of rounds but with an increase in the length of the messages exchanged. It will be shown in next subsections that a decrease in the number of rounds of the protocol is generally much more important than an increase in the bandwidth usage as far as communication delay is concerned.

Table 2
Communication comparison of SAS and SAOTS protocols

	SAS	SAOTS
Number of rounds	3	2
Message length in round 1	$m + h$	$m + (n + 1)h$
Message length in round 2	$m + h + s$	$m + l + s$
Message length in round 3	$m + 2h + s$	–

m: length of message, h: length of random numbers and hash values, l: length of server's statement (if it is employed), s: length of signature, n: number of random numbers used in the OTS.

5.2. How to choose parameters for SAOTS

Before implementing SAOTS, we should decide on the value of various parameters of SAOTS. More specifically we should determine

- the key length of public key signature,
- the length of random numbers and hash values of OTS,
- the value of n and p.

The key length of the public key signature and the underlying components should be selected consistent with the desired symmetric security level for an application. Several security levels have been identified in [24]. For 80-bit security [7] RSA key length should be at least 1024 bits.

Assuming that the hash function used is a secure one, in OTS generation 80-bit length random numbers and hash outputs are good enough for 80-bit security [23].

The value of n and p depends on b, the length of the hash output applied to the message to be signed.

Just like the length of hash outputs in OTS, is it enough to choose the length of hash output applied to the message as 80-bit? In standard public key signatures the length of the hash output is chosen to be 160-bit (e.g. DSA [2] with SHA-1 [11]) to have a security level of 80-bits to 'birthday attacks'. This attack uses the 'paradox' that says that finding two values, x_1 and x_2 which gives the same hash output y on the average requires 2^{80} trials when y has a length of 160- bits.

Therefore the answer to the question really depends on whether birthday attacks are of our concern or not. We think that although it is theoretically interesting, this attack has no practical impact on the security because it is fairly easy to defeat it by simple tricks (e.g. concatenating the 'signing time' to the message before signing it). As a result we can say that b can be chosen as 80 to have a security level of 80-bits. We calculate the smallest n and corresponding p as $n = 84$ and $p = 39$ using Eq. (1). We refer the interesting readers to [26] for the details of birthday paradox and the related attacks.

[7] This level might be considered as the minimum today for strong security since an 80-bit key search would take about seven years using a $10 million machine [25].

5.3. Implementation and experiments

To have a more concrete performance comparison of SAS and SAOTS, we have implemented both of them using MIRACL library [28]. A PC running Windows 2000 with an 800 MHz Pentium III and a 128 MB memory was chosen as the VS and a PC running Windows 95 with a 200 MHz Pentium I and a 32 MB memory was chosen as the clients' machine. Note that today's high-end PDA's and palmtops have a processor speed of 200 MHz. The compiler was Microsoft Visual C++ Version 6.0.

We have conducted two experiments. The first experiment was conducted using a wireless 802.11 b LAN (having a bandwidth of 11 Mbps). Fig. 3 illustrates the experimental setup we have used. The second experiment was over the WAN (Internet) with a very long distance between machines. We have seen that the network delay measurements in the WAN case is not consistent and has a big variance so we decided not to include our results in WAN environment to Table 4 instead we provide a theoretical analysis in the next subsection. Also, see our final note in the end of this section.

To have a reliable communication, TCP was used as the transport protocol. We have chosen a typical message length of 1 KB. The parameters of SAOTS were chosen as described in previous subsection. In addition, the public key e of RSA was chosen to be 65537 since choosing $e = 3$ might cause some security vulnerabilities.

Remember that SAS is a three-round and SAOTS is a two-round protocol. This is why the total delay (the network delay until the signed message is received by the receiver) is smaller in SAOTS in spite of greater bandwidth utilization. Table 3 gives the performance measurements of cryptography primitives on two platforms used and Table 4 summarizes our findings in the experiments.

These experimental results show that SAOTS offers a substantial computational advantage over

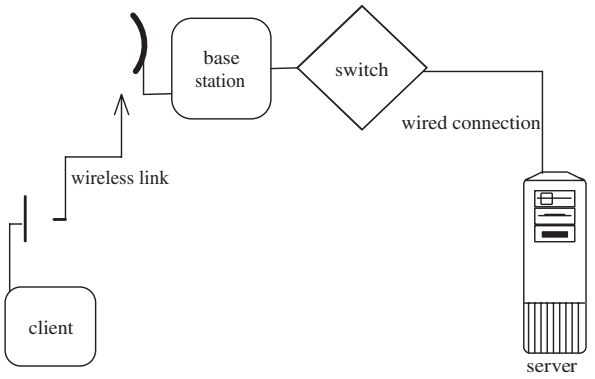

Fig. 3. The experimental setup used for performance comparison of SAS and SAOTS protocols.

Table 3
Performance measurements of cryptography primitives (ms)

	Pentium III 800 MHz	Pentium I 200 MHz
SHS	0.028	0.156
RSA (verifying)	2.220	13.893
RSA (signing)	9.454	59.162
Mapping	0.02	0.1

Table 4
Experimental comparison of SAS and SAOTS protocols (ms)

	SAS	SAOTS
Originator's computation	14.049	0.256
Server's computation	9.482	10.633
Receiver's computation	14.205	14.049
Network delay (wireless LAN)	8.91	6.16

SAS with respect to originator's computation. Moreover, the total time required to send a signed message to the receiver(s) in SAS protocol is at least twice the time in SAOTS (using a wireless LAN).

It is straightforward to see that the gain we obtain using SAOTS will increase if

1. If the protocol is operating in an environment with greater network delays (see the next subsection).
2. A public-key algorithm with a longer key (e.g. 2048-bit RSA) is to be used since as the verification time of public-key signature will increase, only SAS protocol's performance (with respect to computation on originator's machine) will get worse.
3. A public key algorithm (e.g. DSS [2]) where verification of the signature is not more efficient than generating the signature is used.
4. The computational power ratio of the server to a client gets larger i.e. A more powerful server and/or a less powerful client is used.

As a final note, we have also observed that there is a threshold for the extra network delay where signing in a traditional way and sending the signed message directly to the receiver becomes more efficient in terms of total delay of signature generation. The value of this threshold for the network delay is around 45 ms for SAS and around 59 ms for SAOTS. But SAS and especially SAOTS are still preferable in applications where performance of client's computation is a bigger issue or a server is already a built-in element in the application e.g. e-mail, chat over a server, etc.

5.4. Theoretical comparison of network delays

In order to compare the performance of SAOTS and SAS, we have implemented both schemes and measured the time delays in real-time as discussed in previous subsection. Since the implementation (in WAN environment) did not give us a consistent result, it is worth trying to evaluate their delay performance theoretically as much as possible. First of all, we have seen that

the network delay is composed of several delay elements so it can be expressed as a sum of these:

$$\text{Delay}_{\text{network}} = D_{\text{n}} = D_{\text{t}} + D_{\text{p}} + D_{\text{i}} + D_{\text{q}}, \quad (3)$$

where the elements on the right side are transmission, propagation, interface, and queuing delays respectively. The interface delay is the delay on the end hosts i.e. operating system and protocol overhead. Assume that the bandwidth is constant, and the length of the server's statement is equal to length of hash value, then we can rewrite our formula for SAS and SAOTS as follows:

$$\text{SAS} : \frac{3m + 4h + 2s}{\text{BW}} + 3D_{\text{p}} + 3D_{\text{i}} + D_{\text{q3}},$$

$$\text{SAOTS} : \frac{2m + (n+2)h + s}{\text{BW}} + 2D_{\text{p}} + 2D_{\text{i}} + D_{\text{q2}}.$$

Note that in the above formulas, BW denotes the bandwidth of the network and, propagation and interface delays are functions of number of rounds but queuing delay is not. As a matter of fact, one might expect an increase in queuing delay when the packet size increases, but the dependency relation is probabilistic and highly dependant on the underlying network architecture. If we perform a subtraction on these two formulas, we get the following inequality:

$$D_{\text{p}} + D_{\text{i}} + D_{\text{q3}} > \frac{(n-2)h - s - m}{\text{BW}} + D_{\text{q2}}.$$

If this inequality holds, then the network delay is smaller in SAOTS otherwise it is bigger. On the top of the first term on the right side, the extra amount of bits transmitted in SAOTS is given which results in an extra transmission delay (only when the message length is less than 5.5 Kbits). The experiences have shown that in most situations, transmission delay is not the dominant factor of the network delay [27]. (Section 6.6 of [27] is a valuable guideline on performance issues in computer networks in general and most of the arguments there are in favor of SAOTS.)

5.5. How about power efficiency?

Our experiment shows that SAOTS has an advantage over SAS especially with respect to computational requirements on originator's machine. In addition to computational incapabilities, most pervasive devices have another constraint: limited battery life (power).

With respect to power efficiency, we see that increased length of the message to be transmitted (even when we utilize the size reduction technique introduced in the next section) makes SAOTS more problematic. More truly put into words, neither SAOTS nor SAS is more power efficient than traditional public key signing. This is because in wireless networks, transmission consumes much more energy than computation [3]. Therefore SAOTS and SAS are not recommended in settings where battery life is a critical issue (e.g. sensor networks where recharging is not possible or practical).

6. A size reduction technique: SAOTS with hash chains

In SAOTS protocol we have proposed, in the first round the signer sends to the server the message and a one-time signature as well as the public key for the next signature. Is it possible to reduce the size of this bulk?

Yes, it is possible not to send the public key of one-time signature if we utilize the idea of hash chaining. But as we will see, it does not come free.

Now in the initialization phase of SAOTS similar to SAS each user gets a certificate from an offline certification authority for the hash array of length n

$$K_0^k, K_1^k, K_2^k, \ldots, K_{n-1}^k, \quad (4)$$

where n is chosen to be large enough to map the hashed message (e.g. $n = 84$ for 80-bit hash). Each element of the array is the last element of a hash chain of length k where

$$K_j^k = h^k(s_j) = h(K_j^{k-1}) \quad (\text{for } j = 0 \text{ to } n-1). \quad (5)$$

In this equation, $h^k(s)$ is a hash chain of length k and means we apply hash function $h(\cdot)$ k times to an initial input s.

Then the modified version of SAOTS (SAOTS with hash chains) works in two rounds again but

now the originator should receive, verify and store the signature coming from the server:

1. The originator (O) sends m and S^i to VS where
   - m is the message.
   - $S^i = (K^i_{a_1}, K^i_{a_2}, K^i_{a_3}, \ldots, K^i_{a_p})$ denotes the subset of the array of length p that maps the message to an OTS (composed of ith elements of hash chains). The counter i is initially set to $k-1$ and decremented after each run.
2. Having received O's request, VS performs the followings:
   - Whether O's certificate is revoked or not.
   - Computes the mapping of $h(m)$ or in other words finds out which subset S^i would correspond to the OTS of the message.
   - Checks whether for ($q = 1$ to p) $h^{n-i}(K^i_{a_q}) = K^n_{a_q}$ or in a more efficient way ($q = 1$ to p) $h(K^i_{a_q}) = K^{i+1}_{a_q}$ if $K^{i+1}_{a_q}$ has already been received (it depends on the previous message mapping).

If these are all OK, VS signs the message and sends it back to both R and O. For the VS's signature, there are two possibilities:

- VS can sign with a traditional public key signature.
- If in the initialization phase VS gets a certificate for the hash chain array (it should prepare a separate hash chain array for each registered user), VS can also sign using one-time signatures thereby alleviate the verification for those who can not perform public key operations.

After receiving the signed message from VS, both O and R verifies VS's signature. For secure operation, O should sign the next message only after this (off-line) verification. Otherwise, the following attacks can be performed:

- An attacker can generate the (ith) hash elements required to forge the signature on a different message by applying a hash operation on the $(i-1)$th hashes. He then inserts this new signature instead of the O's signature. However the attacker cannot generate a valid signature for any message he wants. This depends on the previous messages signed and the mapping algorithm.
- If VS gets the previous signed message but does not send the signature on this message before he gets the next signed message from O, an attacker cannot forge a signature but now VS can generate a forged signature and that cheating cannot be proven. For O, the signed message will be the only proof for VS's cheating.

Off-line verification and storing of VS's signature before signing another message will be sufficient to avoid these attacks. Since VS has signed the O's signature with the ith hash elements, if an attacker sends a forged signature using the ith hash elements to VS, VS rejects this request (look at the step 2 of the SAOTS protocol above) and the attack is not successful. The VS cannot even generate a forged signature because it has signed the genuine message previously.

Finally, we can say that with a trade of heavier computation requirement and an extra storage requirement in the new version of SAOTS protocol that uses hash chains, the signature length is $p \times h = 3.12$ Kbits (for $n = 84$, $p = 39$ and $h = 80$) and we have a total length save of ($n + 1 - p = 3.68$ Kbits) in round 1 of the protocol.

7. Revocation of public key certificates

Increased use of digital signatures emphasizes the importance of effective and efficient revocation methods so that if a user does something that warrants revocation of his security privileges i.e. he might be fired or may suspect that his private key has been compromised, he should not generate valid digital signatures on any further messages (however, signatures generated prior to revocation may need to remain valid).

In online certificate status protocol (OCSP) [29] (today's state-of-the-art approach to solve the revocation problem) to provide timely revocation information, upon verifier's query a validation server sends back a signed response showing the sender's certificate's current status. The drawback here is that it is impossible to ask a validation

server whether a certificate was valid at the time of signing.

Immediate revocation (the user cannot sign immediately after the revocation takes place) is possible if an online VS is employed. In order to revoke a user's public key, it is sufficient to notify the server. The server maintains a list of revoked users and it rejects signing on behalf of the user if his public key is in the list.

We now want to show a deficiency in the revocation capability of SAS protocol [21]. SAS protocol works in three rounds as explained in Section 2.4. Think of a situation where the user gets the public key signature from the VS in round 2 and postpones the execution of round 3. He then notifies the server to revoke his public key (e.g. claims that his private key has been stolen). Afterwards, he can cheat by executing round 3 and generating a valid signature although his public key has already been revoked.

In our proposal, this deficiency is eliminated since SAOTS protocol works in two steps in opposed to three in SAS.

8. Conclusion and future work

In this paper, we have presented a new efficient verifiable server assisted signature protocol called SAOTS. Server assisted one-time signatures (SAOTS) use a chaining technique where the public key of one-time signature for the next message is attached to the current message before it is sent to the server which then signs it using traditional public key cryptography. It is shown that by this technique, just like proxy signatures generating a public key signature using the signer's (constrained) device is possible without performing any public key operations.

To generate signatures, getting help from a verifiable-server has an advantage over proxy-based solutions since as opposed to proxy-server, verifiable-server's cheating can be proven. Verifiable-server assisted signatures were proposed in the past but they could not eliminate public key operations for the signer. SAOTS protocol implies that for generating a public key signature fully trusted proxy servers are no longer the only option for pervasive devices which cannot perform public key operations by themselves.

Verification transparency is another big advantage of SAOTS over previous verifiable-server approaches. No necessity to store past signatures to prove server's cheating and reduced number of rounds are other benefits of SAOTS. The only drawback of the proposed protocol is the increased length of the message transmitted from the signer to the server's machine.

In recent years researchers come up with other signature schemes which do not base on public-key operations. One of them was the BIBA scheme proposed by Perrig [30]. BIBA's advantages are smaller signature lengths and faster verifying times. Designing and evaluating the performance of a server assisted BIBA signature is a promising future work.

Extending the experimental performance evaluation to other platforms (such as 8-bit microcontrollers) might also be very useful. As another future work, we also plan to compare the energy performance of SAOTS and SAS protocols (as well as traditional signing).

Designing a power-efficient digital signature protocol remains an important open problem. Previous protocols are either computationally heavy as in public key signatures or require long or multiple messages to be transferred as in SAOTS and SAS. The only choice currently available seems to be the proxy signature which has the obvious trust problem.

Acknowledgments

We would like to thank Enver Cavus and Alptekin Cakircali for their help on conducting the experiments. We would also like to thank Albert Levi, Semih Bilgen, and anonymous reviewers for their helpful comments and discussions.

References

[1] R.L. Rivest, A. Shamir, L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) (1978) 120–126.

[2] National Institute for Standards and Technology, Digital signature standard (DSS), Federal Register 56(169), August 30, 1991.
[3] D.W. Carman, P.S. Kruus, B.J. Matt, Constraints and approaches for distributed sensor network security, NAI Labs Technical Report 00–010, 2000.
[4] N. Modadugu, D. Boneh, M. Kim, Generating RSA keys on the PalmPilot with the help of an untrusted server, in: Proceedings of RSA, 2000.
[5] N. Asokan, G. Tsudik, M. Waidners, Server-supported signatures, Journal of Computer Security (November 1997).
[6] K. Bicakci, N. Baykal, Server assisted signatures revisited, in: CT-RSA 2004, The Cryptographers' Track at the RSA Conference 2004, San Francisco, CA, USA, February 23–27, 2004, Lecture Notes in Computer Science, vol. 2964, Springer, Berlin, 2004, pp. 143–156.
[7] K. Bicakci, N. Baykal, Design and performance evaluation of a flexible and efficient server assisted signature protocol, in: Proceedings of IEEE 8th Symposium on Computers and Communications, ISCC 2003, June 30–July 3, 2003, Antalya, Turkey, vol. 2, pp. 1238–1243.
[8] K. Bicakci, N. Baykal, SAOTS: A new efficient server assisted signature scheme for pervasive computing, in: Security in Pervasive Computing, First International Conference, Boppard, Germany, March 12–14, 2003, Lecture Notes in Computer Science, vol. 2802, Springer, Berlin, 2004, pp. 187–200.
[9] S. Even, O. Goldreich, S. Micali, On-line/off-line digital signatures, in: CRYPTO 1989, Lecture Notes in Computer Science, vol. 435, Springer, Berlin, 1990, pp. 263–275.
[10] A. Shamir, Y. Tauman, Improved on-line/off-line signature schemes, in: CRYPTO 2001, Lecture Notes in Computer Science, vol. 2139, Springer, Berlin, 2001, pp. 355–367.
[11] National Institute of Standards and Technology (NIST), FIPS Publication 180: Secure Hash Standard (SHS), May 11, 1993.
[12] R.L. Rivest, The MD5 message-digest algorithm, Internet Request for Comments, April 1992, RFC 1321.
[13] L. Lamport, Constructing digital signatures from a one-way function, Technical Report CSL-98, SRI International, October 1979.
[14] K. Bicakci, B. Tung, G. Tsudik, How to construct optimal one-time signatures, Computer Networks 43 (3) (2003) 339–349.
[15] R.C. Merkle, A digital signature based on a conventional encryption function, in: C. Pomerance (Ed.), Proc. CRYPTO 87, Lecture Notes in Computer Science, vol. 293, Springer, Berlin, 1988, pp. 369–378.
[16] M. Burnside, D. Clarke, T. Mills, A. Maywah, S. Devadas, R. Rivest, Proxy-based security protocols in networked mobile devices, in: Proceedings of the 17th ACM Symposium on Applied Computing (Security Track), March 2002, pp. 265–272.
[17] A. Boldyreva, A. Palacio, B. Warinschi. Secure proxy signature schemes for delegation of signing rights, Cryptology ePrint Archive, Report 2003/096, 2003. Available from: <http://eprint.iacr.org>.
[18] P. Beguin, J.J. Quisquater, Fast server-aided RSA signatures secure against active attacks, in: CRYPTO 95, Lecture Notes in Computer Science, vol. 963, Springer, Berlin, 1995, pp. 57–69.
[19] P. Nguyen, J. Stern, The Beguin–Quisquater server-aided RSA protocol from Crypto'95 is not secure, in: ASIACRYPT 98, Lecture Notes in Computer Science, vol. 1514, Springer, Berlin, 1998, pp. 372–379.
[20] M. Jakobsson, S. Wetzel, Secure server-aided signature generation, in: Proceedings of the International Workshop on Practice and Theory in Public Key Cryptography (PKC 2001), Lecture Notes in Computer Science, vol. 1992, Springer, Berlin, 2001, pp. 383–401.
[21] X. Ding, D. Mazzocchi, G. Tsudik, Experimenting with server-aided signatures, in: 2002 Network and Distributed Systems Security Symposium (NDSS'02), February 2002.
[22] D. Boneh, X. Ding, G. Tsudik, B. Wong, Instantaneous revocation of security capabilities, in: Proceedings of USENIX Security Symposium 2001, August 2001.
[23] R. Gennaro, P. Rohatgi, How to sign digital streams, in: CRYPTO 1997, Lecture Notes in Computer Science, vol. 1294, Springer, Berlin, 1997, pp. 180–197.
[24] National Institute of Standards and Technology, Special Publication 800–57: Recommendation for Key Management, Part 1: General Guideline, Draft, January 2003.
[25] B. Kaliski, TWIRL and RSA key size, RSA Labs, Technical Note, May 6, 2003.
[26] A.J. Menezes, P.C. van Oorschot, S.A. Vanstone, Handbook of Applied Cryptography, fifth ed., CRC Press, Boca Raton, FL, 2001. Available from <http://www.cacr.math.uwaterloo.ca/hac/>.
[27] A.S. Tanenbaum, Computer Networks, third ed., Prentice-Hall, Englewood Cliffs, NJ, 1996.
[28] MIRACL Multiprecision Integer and Rational Arithmetic C/C++ Library. Available from: <http://indigo.ie/~mscott/>. Accessed on September 22, 2004.
[29] M. Myers, R. Ankney, A. Malpani, S. Galperin, C. Adams, Internet public key infrastructure online certificate status protocol—OCSP, RFC 2560, June 1999.
[30] A. Perrig, The BIBA one-time signature and broadcast authentication protocol, in: ACM Conference on Computer and Communications Security, 2001, pp. 28–37.

Kemal Bicakci received the B.S. degree in electronics engineering from Hacettepe University, Turkey in 1996 and the M.S. degree in electrical engineering (computer networks) from University of Southern California, USA in 1999. Between January 1998 and March 2000, he worked on various security projects as a graduate research assistant in USC Information Sciences Institute.

He received his Ph.D. in September 2003 from Informatics Institute, Middle East Technical University, Turkey. He is now a postdoc in Vrije Universiteit, The Netherlands. His

research interests include network security, applied cryptography and medical informatics.

Nazife Baykal received her B.S. degree in Mathematics, M.S. and Ph.D. (1996) degrees in Computer Engineering from the Middle East Technical University (METU) in Ankara, Turkey. Between 1993 and 1994 she worked as a visiting researcher as a NATO science scholar at the Computer Science Department in University of Maryland, Maryland, USA. In 1999, she was appointed as a teaching staff at the Institute of Informatics of the METU. In 2000 she received associated professor title. Between 2002 and 2003 she worked as a visiting researcher in University of Texas, School of Health Information Sciences, Houston, USA. In 2003, Health Informatics Department was established within the Informatics Institute of METU, and she was appointed as the head of the Department.

She has published more than 50 technical papers and 3 textbooks in various areas, including computer networks, health informatics, pattern recognition, neural networks, fuzzy systems and data mining. Her current research interests include computer networks, medical informatics, fuzzy systems, and data mining.