Turk J Elec Engin, VOL.11, NO.3 2003, © TÜBİTAK

Improving the Security and Flexibility of One-Time Passwords by Signature Chains

Kemal BIÇAKCI, Nazife BAYKAL
Middle East Technical University, Informatics Institute,
İnönü Bulvarı, 06531, Ankara-TURKEY
bicakci,baykal@ii.metu.edu.tr

Abstract

While the classical attack of "monitor the network and intercept the password" can be avoided by advanced protocols like SSH, one-time passwords are still considered a viable alternative or a supplement for software authentication since they are the only ones that safeguard against attacks on insecure client machines. In this paper by using public-key techniques we present a method called signature chain alternative to Lamport's hash chain to improve security and flexibility of one-time passwords. Our proposition improves the security because first, like other public-key authentication protocols, the server and the user do not share a secret, thereby eliminating attacks on the server side. Second, from any incorrectly revealed one-time password, unspent passwords cannot be calculated if a signature chain is preferred. Having an infinite length, the chain in our proposition is more flexible and facilitates using the protocol without the complexity of restarting. On the other hand, the disadvantage of signature chain is the longer verification time with respect to hash chain based approaches.

Key Words: *authentication, hash chain, signature chain, public-key authentication protocol, one-time password, network security.*

1. Introduction

Authentication is the process by which a system can determine whether or not a given user is who he/she claims to be. Authentication is the key for information security since if the authentication mechanism is compromised, the rest of the security measures are bypassed as well.

It is widely accepted that authentication uses one or more one of the followings.

- Something you have (smartcards)
- Something you are (biometrics)
- Something you know (passwords)

The last alternative, passwords are the most widely used method for authentication. This is due to its convenience and low cost. While the others depend on specialized hardware devices, this method can be fully utilized by using only software techniques. This is why it is sometimes called "software authentication".

Other than inconvenience and cost, the first 2 alternatives have their own problems. We do not explain these but refer instead to some references [1-3].

The classical way of authentication via passwords in early protocols like telnet is composed of 4 steps:

1. User enters his/her name and password to the client machine.
2. Client machine sends the name and password across the network.
3. Server uses the password to authenticate the user's identity.
4. Server authorizes access for authenticated identity.

Since in this basic protocol the password is transmitted across the network in plaintext (without any encryption), anybody that can intercept the password can use it later for impersonation. This simple attack was the initial motivation behind the design of one-time passwords (OTPs) [7]. At present, to safeguard against this attack, system administrators are switching their computer systems from telnet to secure shell (SSH) [4]. In SSH, where the password is transmitted in an encrypted channel, "eavesdrop and replay" kinds of attack are impossible. This is why at first glance one might think that having the elegant solution of SSH, using OTPs becomes obsolete and is no more than an inconvenience for the user. This claim is not true, as we will show, since the network is not the only place the attacker can steal the password from.

There are several ways in which a password may be snooped directly on the client machine. For instance, someone with root access may maliciously have installed a trojan horse version of an application program or a "wiretap" device driver in the kernel. If the system administrator installing the software is not malicious but careless enough not to check that he has an unmodified version of software distribution, a "keyboard-trapping" routine inside the modified version of authentication software can again capture the password when you are typing. No matter how sophisticated they are, none of the software authentication methods based on traditional passwords can safeguard against the attacks on insecure client machines. Possible countermeasures to this attack are as follows:

1. Making sure that the client machine is secure so that it does not allow someone to snoop the password.
2. We can employ OTPs for authentication purposes. Since each password is valid for only one time, the password, if snooped, is not useful for later authentications.

The first countermeasure is applicable when the user has the full administrative rights of the client machine and all the software programs he installs are reliable. However, this assumption is not realistic in most cases, for instance when the user travels frequently and uses insecure or even hostile clients to login to a server machine.

We reserve the next section for the introductory information on the second countermeasure, OTPs, which offer a viable alternative to traditional passwords, but have some deficiencies with respect to the flexibility and security they provide. These deficiencies are precisely the topic of this paper.

After the next section, this paper is organized as follows. Before going into the details of our proposal, we briefly explain our contribution in section 3. In section 4, to overcome the limitations of hash chain construction used in OTP schemes, we propose the concept of signature chain. In section 5, we show how this new idea can be employed in OTP-s operation. One important issue in using any public-key technique, the issue of securely distributing the public keys is discussed in Section 6, emphasizing the characteristics in our setting. Section 7 summarizes the analysis of our new authentication protocol. We look at the practical issues in section 8. Finally, we end up by providing the results of our performance evaluation study and discussing future works in sections 9 and 10, respectively.

2. One-Time Passwords

OTP schemes, where each password is used only once, offer a viable alternative or a supplement to traditional password schemes. OTP schemes' advantages depend on which mode of operation is used. The modes of operation and corresponding advantages are as follows:

1. In the first mode of operation, to facilitate user-friendliness, each user has only (and should memorize) one password just as in traditional passwords. This password is used to authenticate the user to the client machine (workstation) and then this machine generates the OTP to be sent to the server. On the network connecting the client and the server machines only the OTP is transmitted. Hence this mode, sometimes named as "workstation environment", safeguards only from "eavesdrop and replay" kinds of attack.

2. For applications that require stronger security, it is possible to have the user enter the OTP-s without getting any help from the system. In this mode not only "eavesdrop and replay" attacks are impossible but also the aforementioned attacks on insecure client machines are avoided. Of course, we cannot expect someone to memorize all these OTPs. Now we are in a so-called "human and paper environment" where OTPs are listed on a piece of paper and carried in the pocket[1] . This is an inconvenience for the user, but in most applications demanding a high level of security this inconvenience is tolerable.

Variations of OTP schemes include "sequentially updated OTPs" where there is initially only a single secret password that is shared and the user creates and transmits the new password while he is being authenticated with the previous one, and "shared lists of OTPs" where each user uses a set of passwords each valid for a single authentication and distributed as a pre-shared list [5].

The third alternative, "OTP sequences based on hash chain", is a more elegant design and has more attractive properties than the other 2 variations. First of all, this method is more efficient with respect to bandwidth than sequential updated OTPs. Sequentially updating also becomes difficult when communication failures occur. Second, shared lists have the drawback of list maintainance and distribution.

Hash functions are useful constructions in many cryptographic applications like OTPs and their definition is straightforward; a hash function $h()$ is a function such that

- Given an input string x, it is easy to compute $h(x)$.
- But given a randomly chosen y, it is computationally infeasible to find an x such that $h(x) = y$.

The idea of hash chains was first proposed by Lamport [6] in 1981. Applying the hash function $h()N$ times to a seed will form a hash chain of length N :

$$h^1(s), h^2(s), h^3(s), \cdots, h^{N-1}(s), h^N(s).$$

In the above hash chain, h^{i-1} cannot be generated from h^i by those who do not know the value s.

OTP schemes based on hash chains (Lamport's idea) operate in 3 steps [7]:

[1]Alternatively, mobile devices such as PDAs and cellular phones can be used for storing the OTP list.

Preparatory Step

The server sends a seed in clear text to the user. The user concatenates the password with the seed so that the user can use the password (shared secret) more than once by changing the seed. The result of the concatenation is called "s" on the generation and verification steps.

Generation Step

By applying the secure hash function more than once, a sequence of OTPs (p_i) is produced by the user. The initial OTP is produced by applying the hash function $(h)N$ times

$$P_0 = h^N(s) \tag{1}$$

The next OTP (the first one to be used for authentication) is generated by applying the hash function $N-1$ times

$$P_1 = h^{N-1}(s) \tag{2}$$

and, the general formula is

$$P_i = h^{N-i}(s) \tag{3}$$

By knowing P_i, P_{i+1} cannot be generated.

Verification Step

First of all, the initial OTP P_0 is calculated by the server. Before a user tries to be authenticated, the current value of i and the seed are passed to him/her, so that the user enters the next OTP (The first OTP used for authentication is P_1). The server applies the hash function to the password sent

$$P_i? \equiv ?h(h^{N-i-1}(s)) = h(P_{i+1}) \tag{4}$$

If the output is the same as the one stored on the server side (the previous one), the user is authenticated and the OTP is saved for the next authentication.

3. Our Contribution in a Nutshell

Lamport's idea of hash chains has been widely employed in popular OTP software packages [7] [8], but we are not aware of any further study to improve his idea. Despite the advantages provided by OTPs, they have nevertheless remained on the periphery of security research since their inception. One exception is [9], where the authors extend Lamport's idea to more general access mechanisms by combining it with zero-knowledge techniques.

More recent studies [10] [11] recommend other OTP schemes (not based on hash chains) to overcome the limitations of Lamport's idea but as discussed in the previous section, they have their own limitations.

In [10], the author proposes a new OTP scheme with a shared list of passwords predistributed. Their contribution is to allow the server to decrease the storage requirements by cryptographic techniques. Partially based on our earlier work [12], this paper is the first trying to improve the security and flexibility of Lamport's idea.

Currently in all OTP methods the server stores a secret. For instance in [7], hash chains are constructed from the concatenation of a password and a seed (transmitted to the user in cleartext). The password is the shared secret and needs to be stored on the server side in some way. Even if the password is destroyed after the hash chain is generated (we cannot generate any further chains), the server is susceptible to attacks since the password might be a guessable one and an off-line dictionary attack on the hash chain is very powerful just like in the traditional password scheme (In our research we see that there are various open-source tools to perform these attacks [2]).

Our first contribution is the design of an OTP-based authentication method, which does not need to store a secret on the server side but still has the capability of defending against client-side attacks [3]. Our second contribution is a chain-based scheme in which from any incorrectly revealed OTP, unspent OTPs cannot be calculated therefore, for secure operation our scheme does not assume the user is careful enough to enter the correct password in each authentication (It is obvious that without chaining, with a cost of increased storage, this is easily supported in a shared list of independent passwords). Our last, but not least, contribution is to allow using OTPs without a need to reinitialize the system after a certain number of authentications. By having an infinite length, in contrast to current method in use, the chain in our proposition is more flexible and facilitates using the protocol without the complexity and communication overhead of restarting.

4. Signature Chains

In this section we will propose the SC as an alternative to Lamport's hash chain. Our proposition depends on the idea of public-key crypto-systems. In secret-key cryptography, while we have the same key that is used for both encryption and decryption, public-key cryptosystems involve 2 keys. One of these keys is public, i.e. it is available to everybody. Any message that is encrypted by a public key only can be decrypted back with the corresponding private key. We can also build a digital signature easily by using the public key–private key pair. Any message that is digitally signed with the private key can be verified by using the public key. Since the private key is known only by its owner, only the owner can generate the digital signature. Our proposition uses public-key cryptosystems very similar to digital signature schemes. The only difference is that in digital signatures generally the algorithm is performed on the hashed version of the message, this is because most times the message is too long to be processed by the public-key algorithm. In our case, as we will show in subsection 4.2, we do not need to bother with this kind of pre-processing if we are careful enough in choosing the digital signature scheme to be used.

4.1. Definition

We will define SC as follows:

[2]e.g., http://www.monkey.org/∼dugsong/john-1.6.skey.patch-1 (Last access: September 17, 2003).

[3]An alternative method not to store a secret on the server side is to start the construction of hash chain from a secure random number and to send the last element of this hash chain to the server after digitally signing it. However this method lacks the other benefits of our proposal.

Definition: SC is a chain where the elements in the chain are the signatures obtained by applying the signing algorithm recursively starting with an initial input message[4] .

The length of the chain can be legitimately increased infinitely, thereby facilitating its use without restarting or bootstrapping.

Now we will see how we can construct such a signature chain

Construction: Let algorithm S be a signing algorithm in a signature scheme (e.g., RSA [14], DSS [15]) where d is the private key and V is the corresponding verification algorithm with the corresponding public key e. Let x and y constitute a pair such that

$$S_d(x) = y$$

and

$$V_e(x,y) = \begin{array}{ll} \text{true if} & y = S_d(x) \\ \text{false if} & y \neq S_d(x) \end{array}$$

$S_d^N(x)$ denotes that we apply the signing algorithm S recursively N times to the initial input message (seed) x using the private key d. As seen below, recursive applications results in an (infinite length) signature chain originated from the initial input message x

$$x, S_d(x), S_d^2(x), S_d^3(x), \cdots, S_d^N(x), S_d^{N+1}(x), \cdots$$

4.2. The "Message Recovery" Property

We have previously claimed that in our proposition, unlike hash chains, the previous passwords cannot be generated from the (incorrectly entered) future passwords. In Lamport's hash chain, imagine for instance that instead of the first OTP, the last OTP is entered by the careless user. Then all the previous passwords can be generated from it by applying the (public) hash function successively.

In the SC construction defined above, the attacker cannot generate previous elements from later ones only if the chosen digital signature scheme does not have the message recovery property. We can briefly define this property as follows:

Definition: If the original message can be recovered from the signature itself, then the digital signature scheme has the property called "message recovery" (e.g., RSA [14]).

Now we will demonstrate how an attacker can generate the previous elements of the SC from later elements by employing this property

In RSA, where $n$and b constitute the public key and $a$is the private key, a signature for a message x is composed by

$$y = S_a(x) = x^a \bmod n \tag{5}$$

and the corresponding verification works as follows

[4]We will discuss an alternative method to obtain chained OTPs by signing in subsection 8.1.

$$V_b(x,y) = true \;\; if \;\; x = y^b \bmod n \tag{6}$$

Suppose we choose RSA in constructing a SC, then the first three elements can be constructed as follows by using s as the initial seed

$$P_1 = s^a \bmod n, \quad P_2 = P_1^a \bmod n, \quad P_3 = P_2^a \bmod n$$

Suppose also the user entered P_3 incorrectly instead of P_1 for the first authentication, then an attacker can compute P_2 and P_1 by using P_3 and the public key as follows:

$$P_2 = P_3^b \bmod n, \quad P_1 = P_2^b \bmod n$$

In order to safeguard against this attack, hashing the password before signing it should be performed

$$P_1 = [h(s)]^a \bmod n, \quad P_2 = [h(P_1)]^a \bmod n, \quad P_3 = [h(P_2)]^a \bmod n$$

Then P_2 cannot be generated from P_3 (The attacker can generate $h(P_2)$ but not P_2).

In contrast, DSS [15], which does not have the "message recovery" property, can be used to construct a SC and safeguard against the concerned attack without hashing.

5. One-Time Passwords with Signature Chains

Having defined the SC above, let's see how we utilize this idea in generating OTPs. Suppose the public key e is transmitted to the server securely and s is the seed to be used. Then the first OTP can be constructed by

$$S_d(s) = P_0 \tag{7}$$

When this password is received by the server, it can be verified by applying

$$V_e(s, P_0)? \equiv ?true \tag{8}$$

The general formula for the i^{th} OTP is

$$P_i = S_d^i(s) \tag{9}$$

Or

$$P_i = S_d(P_{i-1}) \tag{10}$$

Note that just like ordinary OTP-s, by knowing P_{i-1}, P_i cannot be generated because d is unknown to the server.

And the verification of the i^{th} OTP is done by applying

$$V_e(P_{i-1}, P_i)? \equiv ?true \tag{11}$$

since P_{i-1} is already received.

In this scheme, unlike Lamport's, the value s does not need to be a secret value.

However for security reasons it also should not be a value freely chosen by the user only[5]. For instance, to agree on the seed value the user can compute the hash of the public-key to generate the seed and the server repeats the same computation to have the same seed to start with. We will summarize the initialization and operation of signature chain based OTP (SCOTP) protocol below.

SCOTP Authentication Protocol

Initialization

i. The user registers onto the server and sends securely his/her public key to the server using one of the methods introduced in the next section.

ii. The server and the user agree on the seed value to be used.

iii. The server keeps a table for each user, which stores the

   1. user's ID
   2. user's public key
   3. OTP sequence number (initially zero)
   4. Previous OTP (initially the seed value)

iv. The user generates an OTP list from the seed value by constructing a SC using his private key. The user can update his OTP list anytime he wishes by computing more elements in the infinite length SC.

Operation:

The steps of the server's operation are illustrated in the following pseudocode:

```
Get user's ID
Find the current value of OTP sequence number of the user and send it.
Repeat
    While the OTP is not entered do
        Wait
    End-While
    Verify the OTP entered
```

[5]For some signature schemes like DSS [15], an attacker can generate a valid signature for a random message. Then the attacker can claim that the random number is the seed to start with. Refer to [5] for more details.

```
    If OTP is correct
        Authentication succeeds
        Update the previous OTP value of the user
        Increment the OTP sequence number
    Else
        Print a warning message
    End-if
Until OTP is correct
```

6. How to Distribute the Public Keys Securely?

We start this section by introducing the security problem in our settings if we do not provide a mechanism to securely distribute the public keys.

A user named Alice has generated a private key–public key pair for herself and let her send the public key to the server. While the public key is transmitted to the server, somebody, let's say Eve changes the public key value into the one which she has generated. The elements in the signature chain to be used for authentication can be generated by Eve but not Alice because Eve is the one who has the private key corresponding to the public key the server thinks Alice has sent. In summary, Eve can impersonate herself on the server because she claims the public key she has generated is Alice's. To safeguard against this attack, an authentic link between the public key and the user needs to be provided. There are various ways to accomplish this task; we will summarize the most popular ones as follows:

1. Using password authentication first: In SSH [4], users are first authenticated with their passwords and then later they generate a public key–private key pair and store this in their account. While his/her private key is protected with the password, the public key is readable to everybody. So the server can read this public key to authenticate the user. However this method is not useful if there is not a password-based authentication method already available.

2. Out-of-band distribution: Similar to agreeing on a password, in the registration phase the user can distribute the public key to the server out-of-band (offline). Alternatively, the public key can be distributed over an insecure channel and only the hash of the public key is sent over a secure channel, such as a telephone.

3. By using digital certificates: A certificate is simply the public key signed by a third trustworthy person usually known as a certification authority (CA). If a public key infrastructure providing protocols, services, and standards in order to employ the certificates securely and effectively, is already available, then our OTP proposal can utilize this infrastructure for secure operation.

4. By "web of trust" model: This model does not require a CA and trust is established at a user level as opposed to higher-level CAs. The "web of trust" model, the last method we introduce in this section, has been popularized by the encryption software PGP (pretty good privacy) [16]. In PGP, the server initially trusts some set of users. Therefore, it will trust the public keys associated with these users. From the initial set of public keys that the server stores in a key file, transitive trust may be used to trust other public keys. For example, if the server trusts Alice in its key file and Alice trusts Bob, the server can choose to trust Bob and add Bob's public key into its key file.

7. Analysis of the Proposed Authentication Method

We have already shown why OTPs are preferable when insecure client machines are used. Our proposition, OTP with SC, has some other useful features. These are

1. Since the signature chain is constructed by public-key techniques only, no secret is shared between the client and the user just like other public-key authentication protocols. This important property eliminates the possibility of attacks on the server side (It was previously proved that the system also protects the user from the client side attacks in contrast to other software authentication methods based on public keys).

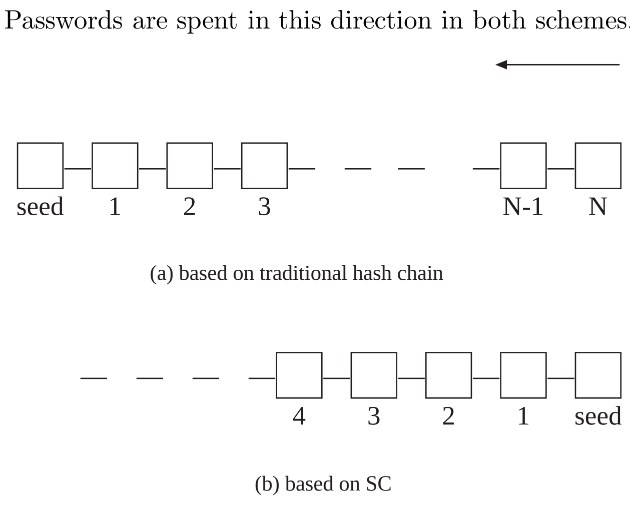
Passwords are spent in this direction in both schemes.
seed
1
2
3
N-1
N
(a) based on traditional hash chain
4
3
2
1
seed
(b) based on SC

Figure Comparison of one-time password schemes

a. OTPs based on hash chain are spent starting from the N^{th} element in the chain and sufficient for N authentications at most.
b. OTPs based on SC are good for infinite number of authentications.

2. By employing a signature scheme, which does not have the "message recovery" property, it was shown that from any incorrectly revealed OTP, unspent OTPs could not be calculated; therefore, for secure operation our scheme does not assume the user is careful enough to enter the correct OTP in each authentication.

3. The figure shows the comparison between OTP schemes based on hash chain and the SC. In this figure, each rectangle demonstrates an OTP and the line connecting these rectangles shows the hash operation for the hash chain and signing operation for SC, respectively (each starts from the seed value). As seen, we have much more flexibility in using the SC, whereas in Lamport's OTP scheme the user will be

able to be authenticated by the system at most N times. For the (N+1)th authentication, the system should be restarted or, in other words, the preparatory step should be repeated [6]. If the computer used to generate the OTPs was not connected to the network for security reasons, to get the new seed value from the server the connection should be re-established.

By using SC, users can generate OTPs in any number they wish in their secure (home) machine and print them out on a piece of paper. These OTPs will be used later to be authenticated from insecure client machines. Should the use be paranoid, to protect the private key used in generating the OTPs the user can disconnect his/her computer from the network.

There is one attack left called an active network attack, which is both effective on OTP protocols with or without a SC. An example of this more sophisticated attack is a "hijacking connection" attack, in which the connection is taken over after the user has authenticated himself to the server. One possible countermeasure to this attack will be explained in subsection 8.3.

8. Practical Considerations

In this section, we will look at 3 issues that are considered to be important for putting our new authentication method into practice.

8.1. Signature Chain in workstation environment

As stated in section 2, there are 2 modes of operation in OTP schemes. As opposed to a human and paper environment, in a workstation environment, the workstation calculates the OTP on behalf of the user after the user enters his/her traditional password to the system. This is a more convenient approach since the user does not need to carry an OTP list.

In this mode, other than the attacks on insecure client machines, an attack called small n attack [13] is of concern. In this attack, after the attacker impersonates the server, he/she asks the workstation for a future OTP (with a smaller value of n) so that he can generate a list of valid OTPs by using the hash chain's one-way property. Signature chains are also effective to safeguard against this kind of attack because from any incorrectly revealed OTP, unspent OTPs could not be calculated.

In a workstation environment (as well as in a human and paper environment), it is also possible to design the protocol so that the server asks for an OTP with a randomly determined sequence number of n. The goal here is to make the attacker unable to guess when an incorrectly revealed password is useful for impersonation. However, if this design is preferred in a workstation environment, we have now a more serious performance problem than the one that will be mentioned in the next section. Suppose that for instance the server asks for the hundredth OTP and currently the tenth OTP is stored on the workstation. Using the proposed SC construction, the workstation should perform 90 public key operations in real time, which is of course a burden for operation. For this problem to generate OTPs, we propose the following alternative method, what we call a counter method.

1. The server and the user agrees on the seed value (designated s) just like in our original proposal.
2. However, in this method if the server asks for the hundredth OTP, the workstation computes the signature on the value of $(s + 100)$ and returns this signature as the OTP.

[6] [5] For instance, in S/Key system a special command "keyinit" should be executed [7].

Notice that in this method only one public key operation is required.

There is an interesting analogy between the 2 alternatives we have proposed to generate OTPs using public-key cryptography and the modes of operation in block ciphers (secret key cryptography), which specifies how a block cipher can be extended to process messages of arbitrary length [1]. While the original signature chain construction we have proposed is the analog to output feedback mode, the method we proposed in this subsection is very similar to the counter encryption mode of block ciphers. We refer interested readers to [1] for information on modes of operation of block ciphers.

8.2. Incorrectly revealed passwords in SCOTP protocol

As we have stated, in SCOTP protocol an attacker cannot generate unspent OTPs from the incorrectly revealed one. However, there is still the risk to use the revealed OTP for impersonation if the attacker can correctly guess when the OTP at hand is valid. If the proposed SC construction is used, it is easy to see that an attacker has no choice other than blindly trying the revealed OTP from time to time; however, in the counter method we observe that the attacker can find out the sequence number of the OTP at hand. As a result, it is evident that SC construction is more secure than the counter method when incorrectly revealed passwords are concerned.

To eliminate the risk of revealed OTPs totally, the SCOTP protocol can be extended so that the user can choose to enter more than one password at any time. Now the user, aware of revealing the password, can enter the incorrectly revealed password just after the previous OTP in a single authentication so that the OTP intercepted by the attacker is useless forever (The next OTP after the revealed one in the list is required for the next authentication).

8.3. Hybrid usage

It is possible to combine SSH [4] and OTPs authentication methods to benefit from the strengths of both. Session hijacking is one of the active network attacks in which OTPs cannot cope. So to protect the user from these kinds of attack, SSH can be used with OTPs. Another reason to use such a combination might be the need for confidentiality of the data exchanged.

By routine checks for malicious software, it is possible to secure the machine we have the root privileges to; therefore, we may choose not to use OTPs and only use SSH. So the inconvenience of using OTPs is left to other machines that are not trusted.

9. Performance Evaluation

To compare the performances of signature chain and hash chain, we have conducted an experiment on a PC with an 800 MHz Pentium III and 128 MB memory. The library used was MIRACL version 4.7 [7] and the compiler was Microsoft Visual C++ Version 6.0. SHS [16] and DSS [15] with a key length of 1024 bits were chosen to generate the hash chain and SC, respectively. Our implementation results show that using public-key algorithms instead of hash functions does result in a degradation of the performance in verification of OTPs (hash functions are designed to be very fast, only 0.028 ms is sufficient to perform one hash operation). However we claim that verification is sufficiently quick (around 6 ms) and does not produce

[7] Available at http://indigo.ie/∼mscott/, (Last access: September 17, 2003).

a significant problem.[8]

We also measured the time required to sign (to generate an element of the SC) and found that it is slightly longer than 5 ms. In the "human and paper environment", the client can generate OTPs and print them on a piece of paper before the operation. Therefore, although the timing for SC is comparable high with respect to hash chain, we think that this does not create a bottleneck in the operation since the computation is performed offline (not in real-time).

10. Conclusion and Future Work

In this paper, using public-key techniques we have provided a method called SC as an alternative to hash chain to improve the security and flexibility of OTPs. More specifically, SC is preferable in the following application scenarios that compensate the additional delay of OTP verification in an SC-based OTP (SCOTP) protocol:

- When storing a secret on the server side should be avoided to safeguard against attacks on the server machine.
- When a chain-based OTP method is desired, which does not assume that the user is careful enough to enter the correct OTP in each authentication (Previous chain-based solutions have the problem of that if the attacker can see one of the last passwords on the list, then all previous passwords can be calculated from it).
- When the flexibility of SC, which facilitates using OTPs without the complexity and communication overhead of restarting and results in a more user-friendly OTP solution, is a required feature.

The use of wide-ranging authentication services based on public-key cryptography, including the one we have presented in this paper, becomes practical if it is complemented by a trustworthy means to manage and distribute the public keys. We believe that the SCOTP protocol will be more useful when public key infrastructure deployment reaches its true potential. As a future work, it is promising to implement a fully functional OTP scheme using the SC idea and integrate it with SSH for hybrid operation. Another future work is to experiment with different signature schemes to make the operation of our protocol more efficient.

References

[1] R. Anderson, Security Engineering: a guide to building dependable distributed systems, Wiley Computer Publishing, 2001.

[2] Bruce Schneier, Secrets and Lies: digital security in a networked world, Wiley Computer Publishing, 2000.

[3] R. Anderson, M. Kuhn: Tamper Resistance – a Cautionary Note, the Second USENIX Workshop on Electronic Commerce Proceedings, Oakland, November 1996.

[4] T. Ylonen. SSH - Secure login connections over the Internet, In Proc. of 6th USENIX Security Symposium, July 1996.

[5] A. Menezes, P. Van Oorshot, S. Vanstone, Handbook of applied cryptography, CRC Press, 1996.

[8] The server that verifies OTPs is generally a more powerful device. Similar to our proposal, popular authentication protocols like SSH use public key cryptography and have similar figures for the timing.

[6] L. Lamport, Password authentication with insecure communication, Communications of the ACM, 24(11), November 1981.

[7] N. Haller, The S/Key One-Time Password System, Proceedings of the Symposium on Network & Distributed Systems Security, Internet Society, San Diego, CA, February 1994.

[8] D.L. McDonald, R.J. Atkinson, C. Metz "One-Time Passwords in Everything (OPIE): Experiences with Building and Using Strong Authentication," In Proc. of the 5th USENIX UNIX Security Symposium, June 1995.

[9] D. de Waleffe, J.J. Quisquater: Better login protocols for computer networks. In Proc. of ESORICS, France, October 1990.

[10] A.D. Rubin, Independent One-Time Passwords, USENIX Journal of Computer Systems, February 1996.

[11] M. Kuhn, A One Time Password Login Capability, http://www.cl.cam.ac.uk/∼mgk25/otpw.html

[12] K. Bicakci, N. Baykal, Infinite Length Hash Chains and Their Applications, Proc. of IEEE 11th International Workshops on Enabling Technologies (WETICE 2002), June 2002, Pittsburgh, USA.

[13] C. Kaufman, R. Perlman, M. Speciner, Network Security, Private Communication in a Public World, Prentice Hall Series, Second Edition, 2002.

[14] 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.

[15] National Institute of Standards and Technology (NIST), FIPS Publication 186: Digital Signature Standard (DSS), May 19, 1994.

[16] P.R. Zimmermann, The Official PGP User's Guide, MIT Press, Cambridge, Mass., 1995.

[17] National Institute of Standards and Technology (NIST), FIPS Publication 180: Secure Hash Standard (SHS), May 11, 1993.