# Single Sign On Architecture with Dynamic Tokens

Fumiko Satoh, Takayuki Itoh
IBM Research, Tokyo Research Laboratory
1623-14, Shimo-tsuruma, Yamato-shi, Kanagawa, Japan
{sfumiko, itot}@jp.ibm.com

## Abstract

*Single Sign On (SSO) is a useful technology that allows users to skip bothersome authentication processes during accesses to multiple services. It is particularly useful for services for mobile terminals because of their limited resources and interfaces. Some existing SSO mechanisms only verify static data such as IDs and passwords. However, we consider that it will be quite useful if they could deal with dynamic data. We propose a new SSO architecture that uses "Dynamic Token"s that describe dynamic data such as a payment history. The architecture introduces an additional server, named "Circulator", which manages the latest token values to service providers. Accordingly, the providers can correctly verify the token values sent from clients. This paper proposes an efficient algorithm for Circulator to effectively visit the providers. The result of our experiment shows the efficiency of the algorithm.*

## 1 Introduction

Nowadays, various mobile services can be accessed via the Internet. In many current systems, providers usually have independent client authentication systems and data is not shared among them. When a client is going to access multiple services, it has to make registrations appropriately and chooses a proper identification to access each service. In particular, the mobile terminals are not easy to input identification information many times. Hence clients might feel such independent authentication is bothersome. Additionally, if a client wants to use a commercial service, an extra payment process is required. A Single Sign On (SSO) architecture can solve the problem where a client needs to have several identifications for each service. Some SSO architecture have became common already, such as Microsoft Passport [1]. In these architectures, it is used a static token, such as IDs and passwords, for client authentication. Besides, it needs an external process when service providers require information for payments, such as credit card numbers. For inexpensive services, such as "pay per click" or "pay per view", the micropayment systems may be preferred over credit card systems. PayWord [2] is one of the current micropayment architectures, and it is very useful comparing it to a credit card payment. The original PayWord architecture is only available for a specific service and cannot be used with SSO. The extended PayWord architecture [3] has been proposed which can be used among several services. However it is not suitable because it needs many digital signatures, which are too costly for mobile terminals. This paper proposes a new SSO architecture with "Dynamic Token"s which simultaneously deal with the authentication of clients and the verification of their dynamic data such as payment histories. Additionally, an additional server named "Circulator" is introduced behind providers. Consequently, the perfornace of a client authentication process is enhanced comapred to a current SSO architecture. The basic framework of this architecture is shown in Figure 1.

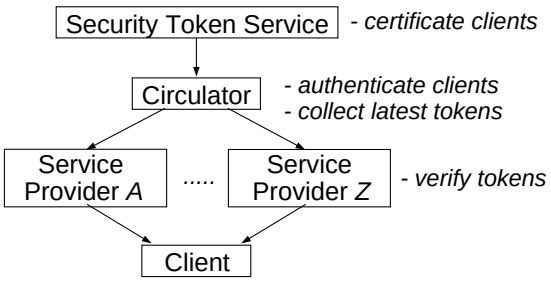


**Figure 1. The basic framework of the proposed architecture**

The particular terms in this paper are defined as follows:

- Dynamic Token: A security token which is issued by Circulator. It is used for the client authentication and verification. Circulator issues a Dynamic Token as a sequence of token values ($D_1, D_2, ..., D_n$), and a client

uses these values in sequential order. Each token value can be used only one time, so it is called a Dynamic Token. It is necessary to know the value of $D_{n-1}$ to check whether the token $D_n$ is valid.

- Circulator: An additional server which is placed behind providers. Its roles are issuing Dynamics Tokens for clients, collecting used token values from providers, and sending the history of used token values to providers.
- Security Token Service: A third party certification organization. Circulator asks it to provide certificates of unknown clients.
- Certification: Issue of a certificate, such as an X.509 Certificate. We assume that Circulator trusts this certificate.
- Authentication: Validation of clients. When the first access from a client, it is authenticated by verifying the signature on the token value with the client certificate.
- Verification: Validation of a current token value. In this architecture, it is used as the alternative of the client authentication after the client is once authenticated.

In this paper, Section 2 describes the proposed architecture, and Section 3 shows an efficient circulation algorithm.

## 2 New SSO architecture

### 2.1 Outline of the proposed architecture

As mentioned in Section 1, the proposed SSO architecture uses a Dynamic Token, which is a sequence of token values ($D_1, D_2, ..., D_n$). As a result, the authentication of clients and the verification of the token values are simultaneously deal with.

This proposed architecture introduces Circulator. The main roles of Circulator are issuing Dynamic Tokens for a certificated client by the Security Token Service and circulating among providers. Visiting providers by Circulator, it collects the histories of used token values and sends the latest used token value for each client to providers. The latest used token value is then used for the next client authentication by the providers.

We assume that a client contacts Circulator to issue its Dynamic Token in the first place. If Circulator does not have a certificate for the client, it asks to the Security Token Service to get a client certificate such as an X.509 Certificate. Receiving certificate and trusting the client, Circulator issues a Dynamic Token, sends it to the client and caches it for itself.

When the client accesses to a provider, it sends the first token value $D_1$ with the signature. The provider sends $D_1$ to Circulator for requesting the client authentication. Here, Circulator should cache the certificate and the Dynamic Token of the client. Circualtor can authenticate the client by verifying the signature on $D_1$ with the certificate, and verify $D_1$ by using the cached Dynamic Token. When the client authentication is finished, the circulator sends the client certificate to the provider. The the provider evaluate the obtained certificate whether the client is allowed to receive a service. Finally, the provider can provide the service to the client and caches the certificate. The client signature is only required at the beginning. Because the token value of the client and the certificate can be mapped in the first access, the provider can authenticate the client by itself in the next.

Due to the cache, the provider can verify the client token value $D_2$ by itself when the client accesses the same provider again. Hence we can assume that the client can be authenticated by the verification of the token value. In this case the provider can respond without establishing an extra connection to Circulator, shown as "Model 1" in Figure 2. Even if a client accesses provider B with $D_3$ after accessing provider A with $D_2$, provider B can authenticate the client by itself if it knows a previous used token value $D_2$.

On the other hand, there are some cases that the provider should connect to Circulator to ask for the client verification and authentication. One of the cases is shown as "Model 2" in Figure 2. This case may happen when the used token values cached by the provider might be obsolete, or the client might tamper its token value. If the authentication and verification by Circulator fails, it is required three connections for the client authentication, shown as "Model 3" in Figure 2. In this case, there are two possible situations: Both used token values cached by the provider and Circulator are obsolete, or the client might tamper its token value. Since Circulator knows which provider is holding the latest used token value, it can collect the latest one from that provider. Using this collected token value, Circulator can authenticate the client at last.

To be applied "Model 1" in as many cases as possible, Circulator visits providers to collect and send the latest token values for each Dynamic Token. It may be possible to occur "Model 3", but we found that 'Model 3" was very rare in our experiments.

### 2.2 Applying to a payment process

By using PayWord [2] as a Dynamic Token, the proposed architecture realizes SSO with a micropayment mechanism. PayWord is a set of sequential numbers generated by a one-way hashing function. It can be used as a Dynamic Token: $D_1$=$W_0$, $D_2$=($W_1$, 1). If the provider knows $D_i$ for the client already, the provider can verify the client which accesses with $D_{i+1}$. When a provider has already received a client certificate and authenticated with the signed token value, then it is possible to replace the authentication of the client by the verification of the token values. The payment

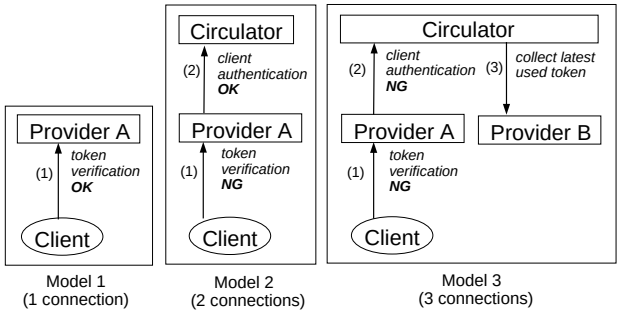


**Figure 2. Three models of the proposed SSO**

histories by PayWord are treated as the used token values in this case.

The combination of the proposed architecture and PayWord realizes a reasonable system for micropayment and SSO for mobile terminals. PayWord is suitable for mobile terminals because of using a simple one-way hash function, which is much faster than the signature process. Also, this architecture realizes the service by "Model 1" in many cases, hence it saves the time to get services for clients.

Applying PayWord, we need to consider the security properties against attacks by a third party. It is described as follows:

- Eavesdropping: Even if the token value is stolen during the connection, the stolen token cannot be used the next time. The thief cannot calculate the next value by itself.
- Tamper: Even if a tampered token value is sent to a provider, the provider will recognize it when the verification of the token value fails.
- Spoofing: The same sequence of a Dynamic Token cannot be generated by a third party. Also, a spoofing client cannot be authenticated because it cannot generate the proper signature value in the first access. When a spoofing client accesses with a stolen token, this corruption can be found by Circulator. However, it is impossible to identify the spoofing client. This property depends on PayWord, it is no matter for the micropayment.

The security property against a client's corruption is as follows:

- Using the same token values with two providers: If Circulator has not completely finished sending to the latest used token values to the providers, the client might use the same token with multiple providers. Even if the client can get service at that time, this corruption will be tracked after Circulator collects the histories of the used token values. The provides can charged the client for the extra payments.

This paper assumes that Circulator and providers trust each other, and the providers' corruptions is not be considered.

# 3 Circulation algorithm

## 3.1 Rules of the circulation algorithm

Another architecture is possible for SSO using Dynamic Tokens by introducing a proxy server between clients and providers [4]. The proxy server can verify the token values sent from clients, before connecting them to providers. We call the architecture a "proxy-based" architecture in this paper. This architecture always needs two connections: the client to the proxy, and the proxy to the provider. An advantage of our architecture against a proxy-based architecture is the smaller number of connections for a client. It is archived by using the efficient circulation algorithm described in this section.

Even if Circulator uniformly visits providers in order, our experiment shows that the new architecture needs less than two connections for the client authentication on average. It means that it is better than a proxy-based architecture. However, if Circulator visits the providers using intelligent algorithms, the providers can effectively share the histories of token values. As a result, the number of connections can be further reduced. Circulator calculates scores for each provider based on certain rules. It chooses the provider that has the highest score to visit. This scoring rules are important to effectively share the latest token values among multiple providers using this architecture.

There are two rules of the circulation. The first rule is visiting the providers which have not been visited for a long time (Rule#1). The second is visiting popular providers which may have a lot of the latest token values (Rule#2). We applied scoring functions based on these rules as follows. The score of provider $i$, $S_i$, is calculated by the function: $S_i = a \times \Delta t_i + b \times m_i$. Here, $\Delta t_i$ is the time during which the provider has not been accessed by the circulator, applied according to Rule 1. The value $m_i$ is the number of accesses to the provider $i$ from all clients, applied according to the Rule 2. Circulator basically calculates the scores of each provider for every visit, however, several variations are possible. For example, Circulator can choose some providers at the same time, and visit them without calculating the scores again.

## 3.2 Algorithm experiment

We implemented and tested the scoring algorithm as follows. In our test scenario, we assumed multiple clients which have various characters, multiple providers which have various types of services, and one circulator. The details of the assumption in this experiment are described

| | Circulator frequency | | | | |
|---|---|---|---|---|---|
| | 2.5% | 5% | 10% | 15% | 20% |
| uniform | 2.50 | 2.31 | 1.91 | 1.41 | 1.17 |
| score-based | 2.34 | 1.99 | 1.52 | 1.25 | 1.17 |

**Table 1. The results of the average number of connections in "uniform" and "score-based"**

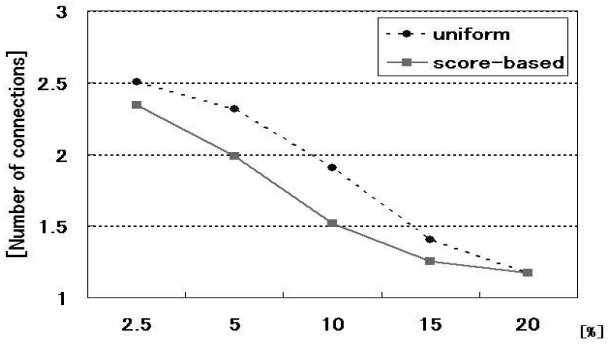


**Figure 3. The results of the average number of connections in "uniform" and "score-based"**

in [5]. Our experiment counts the number of connections from clients to providers for each client authentication process. Note that this proposed SSO architecture is suitable for mobile terminals, hence the performance from the mobile clients is the most important topic. It is differ from the whole performance of this architecture. This experiment confirms that the proposed architecture requires fewer connections from the client compared to the existing proxy-based architecture. The total number of connections which is included that of the providers and Circulator, might be increased, although, this is out of the scope this paper.

The parameters in our experiment is as follows: 200 clients, 30 providers and each client makes 50 requests to various providers. We tested using two parameter sets in the scoring function. One set of parameters is where $a = 1$, $b = 0$, named "uniform". It means that Circulator uniformly visits the providers. Another set is $a = 0.5$, $b = 0.5$, named "score-based". In this case, Circulator visits high-score providers more frequently. We tested five frequencies for Circulator. The frequencies are described as the ratio of the number of Circulator visits to providers, to the average number of client accesses to the providers. We used the ratios of 2.5%, 5%, 10%, 15%, and 20%. We tested the proposed architecture with the above parameters, and compared the average number of connections. The average numbers of connections in this experiment are shown in Table 1 and Figure 3. The experiment shows that the average number of connections was reduced to less than 2 when Circulator visit frequency. In these case, the proposed architectre is better than the proxy-based architecure. The average numbers between the configurations "uniform" and "score-based" are almost equal and very close to 1 in the case that the Circulator frequency is 20%. Even if the frequency of Circulator visits is not enough, the "score-based" configuration shows better result. The result of our experiment clearly shows that the effective configuration of the circulation is very important.

This architecture performance is very dependent on the character of the services. This architecture will achieve better performance than the current architecture for a service which is often accessed by a specific client, such as mail services,

## 4 Conclusion

This paper proposed a new architecture of SSO introducing Dynamic Tokens and Circulator. It makes possible to reduce the number of connections for client authentication. The paper also proposed a scoring algorithm so that Circulator effectively visits providers to manage the Dynamic Tokens. Our experiment confirmed that the algorithm and its effective configuration contributed to reduce the average number of connections. The following issues are possible future works of our study:

- Experiments with larger numbers of clients and providers.
- Optimization of parameters in the circulation algorithm.
- Consideration of multiple Circulators for performance improvement.

## References

[1] Microsoft. .Net Passport. http://www.microsoft.com/net/services/passport/.

[2] Ronald L. Rivest and Adi Shamir. PayWord and MicroMint: Two Simple Micropayment Schemes. In *Security Protocols Workshop*, pages 69–87, 1996. http://citeseer.nj.nec.com/rivest96payword.html.

[3] Manho Lee and Kwangjo Kim. A Micropayment System for Multiple-Shopping. http://citeseer.nj.nec.com/lee02micropayment.html.

[4] Japanese patent. JP2002-288139A.

[5] Fumiko Satoh and Takayuki Itoh. Single Sign On Architecture with Dynamic Tokens. IBM Research, TRL Research Report, RT0542.