

Pervasive Authentication Domains for Automatic Pervasive Device Authorization

Reiner Sailer
IBM T. J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532
Sailer@watson.ibm.com

James R. Giles
IBM T. J. Watson Research Center
19 Skyline Drive, Hawthorne, NY 10532
GilesJam@watson.ibm.com

Abstract

In a pervasive computing environment, users have many devices that are used to initiate or answer remote service requests, such as obtaining real-time stock quotes, handling corporate e-mail, or accepting telephone calls. We envision that in the future, many applications will be distributed, running across many of a user's specialized pervasive devices rather than on a single system. In this case, a user needs the ability to "log into" the personal pervasive domain which spans each of the pervasive devices representing this user. In addition, the pervasive devices belonging to the user's pervasive domain must be able to represent this user to external services. We solve the problem of managing the authorization for pervasive devices belonging to a user's personal pervasive domain by introducing a central personal authorization gateway that accompanies the user and allows pervasive devices in the user's pervasive domain to be automatically configured and authorized.

1. Introduction

It is becoming increasingly common for individuals to operate many devices that have the ability to connect to communication networks. In particular, some individuals carry many electronic devices such as personal digital assistants (PDAs), laptop computers, wireless telephones, sensors [9], or digital watches [5]. that can all be used to communicate or access information over wireless or wireline communication networks. We call such devices that a user carries pervasive devices. In many cases, communication with these pervasive devices needs to be done in a secure manner to ensure confidentiality and integrity of data, as well as to protect the services from unauthorized use.

This need for security places a great burden on users because they must provide authentication and authorization credentials for each device used for secure communications. The problem is compounded by the fact that many devices,

such as digital watches, do not have convenient user interfaces for entering credentials.

2. Pervasive Authentication Domains

Pervasive Authentication Domains allow pervasive devices to be automatically authorized to represent a user by eliminating the need for multiple authentications and by allowing pervasive devices to seamlessly and securely join the authorization of their user. Authorization is restricted by security assertions that protect users from abuse of stolen, lost, copied, or otherwise compromised pervasive devices.

A Pervasive Authentication Domain consists of a device for managing a user's domain, called the Personal Authentication Gateway, and the other pervasive devices which are considered to be inside the domain. Device membership in the Pervasive Authentication Domain is managed by the Personal Authentication Gateway, and depends on factors such as the proximity of the pervasive device to the gateway and the configuration of the domain.

The Personal Authentication Gateway is configured with the user's credentials for external services and a set of pervasive devices which are eligible for membership in the Pervasive Authentication Domain. When the user logs into the Pervasive Authentication Domain, the domain will provide credentials automatically to the other pervasive devices in the domain when they need to communicate securely on behalf of the user. This automatic authentication provides significant benefit for users who have many pervasive devices such as PDAs, wireless phones, and laptop computers.

The Personal Authentication Gateway is assumed to be powerful enough to protect the contents stored in it through power-on passwords and powerful enough to encrypt long-term credentials. We protect long-term secrets in the gateway and only deliver short-term credentials derived from these secrets to the less-protected pervasive devices. We do not require the Personal Authentication Gateway to perform complex private and public key operations; rather we assume short-term credentials are created from challenge-

response authentications based on symmetric secrets. The pervasive devices are assumed to be computationally weak and relatively insecure, so important features of our system are that pervasive devices only receive short-term secrets and credentials are communicated between the gateway and the devices via lightweight protocols. The protocols employ only symmetric encryption and hash computations on small data blocks coupled with infrequent credential updates, so they are computationally inexpensive.

2.1. Architecture

The Pervasive Authentication Domain consists of a Personal Authentication Gateway and a collection of pervasive devices. The Personal Authentication Gateway is transparent to external parties and constitutes the security hub for the domain. A pervasive device can request its security configuration at boot-time from the gateway or the pervasive devices can refresh their security configuration on demand. The architecture of the Personal Authentication Gateway and pervasive devices that implement the Pervasive Authentication Domain is illustrated in Figure 1.

Pervasive devices that have membership in the domain communicate with the Personal Authentication Gateway and external services over networks which can be wireless, wireline, or through direct communication as in infrared. The Personal Authentication Gateway has a Token Server component which responds to token requests from Token Client components. Each pervasive device has a Token Client component which obtains user authorization credentials when a client application on the pervasive device needs to communicate with an external service. The Personal Authentication Gateway holds the user tokens in a Token Store and stores the configuration in a Policy component indicating the devices with membership in the domain and the circumstances under which the Token Server will respond to a Token Client.

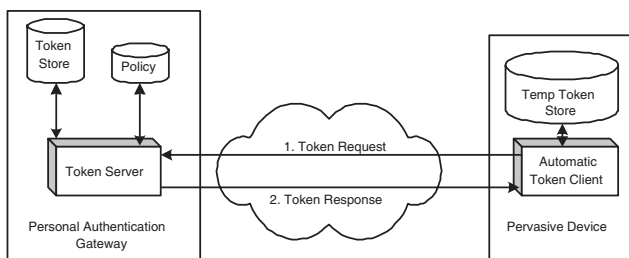


Figure 1. Functional Block Diagram

There are three main scenarios in which a pervasive device obtains tokens from the Personal Authentication Gate-

way. The first of these scenarios is at power-on or boot time, when the device pulls tokens from the gateway that enable relatively long term communication among the various devices in the domain. This scenario is similar to DHCP [2] for network configuration.

The second scenario is pulling tokens on demand to access an external service. This scenario is typically used to satisfy a challenge-response authentication (e.g., CHAP [12]) from external services. Figure 2 illustrates this scenario, where a pervasive device tries to access an external service and receives a challenge from the service as part of a challenge-response authentication. Afterward, the pervasive device sends a Token Request containing the challenge to the Personal Authentication Gateway, which computes the response and sends it back to the pervasive device as part of Token Response. The pervasive device then sends the response to the service and gains access, at which time the service may choose to set a cookie [8], which will allow repeated access for a limited time. In this way, the pervasive device gains access to a service on behalf of a user, without knowing any long-term secrets that could become compromised if the device is lost or stolen.

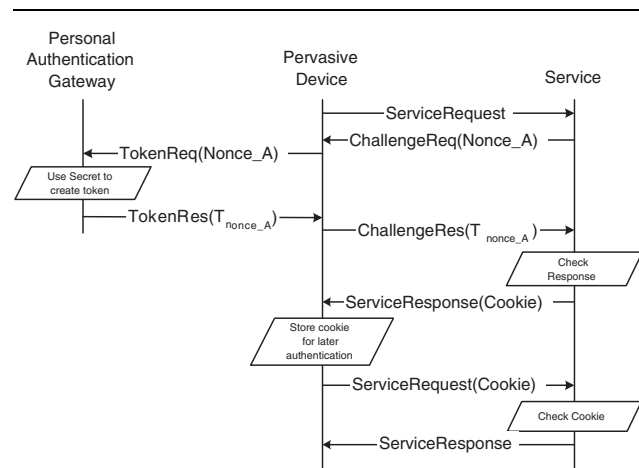


Figure 2. Message Flow Example

The third scenario is a token push initiated by the Personal Authentication Gateway when a token or configuration needs to be reset or when the profile of the device needs to be changed. This scenario is similar to the other scenarios, except that it is initiated by a broadcast from the gateway, which is then followed by the regular token pull model. Because the pervasive device may not always be able to communicate with the gateway, the token push scenario is considered best-effort and unreliable.

2.2. Automatic Pervasive Authorization

Registration. Before a pervasive device can participate in a Pervasive Authentication Domain, it must be registered with the domain. To register, the Token Client of a pervasive device sends a registration request to the Personal Authentication Gateway, which can include the identification of a particular domain (DomainID) that the device wants to join; otherwise a default domain is assumed. The gateway chooses a unique slave identification (SlaveID) for each pervasive device in a chosen domain. This unique identification can be constructed in part from a unique hardware identification for the pervasive device. The gateway also computes a slave secret (SlaveSecret) which is a SHA1 Secure Hash [4] of a master key known only to the gateway, the SlaveID, and the DomainID. The SlaveID, DomainID, and SlaveSecret are communicated to the pervasive device with the user's permission, either with manual entry by the user on the pervasive device, or using an automatic mechanism.

To provide automatic distribution of registration information, we use a shared password to securely communicate the information (SlaveID, DomainID, and SlaveSecret) between the gateway and the pervasive device. This is accomplished by entering the same shared password at the gateway and the target pervasive device at registration time. Alternative mechanisms for communicating the values are SSL links (protected by Client and Server certificate authentication) between the pervasive device and the Personal Authentication Gateway, or removable storage devices such as a Universal Serial Bus memory stick.

Authorization Protocol. The authorization protocol is started whenever a client application on a pervasive device needs to communicate with an external access-controlled service on behalf of the user. The Token Client determines its DomainIDs and corresponding SlaveIDs and SlaveSecrets for the Pervasive Authentication Domains to which it belongs. These identifications and secrets are obtained by the Token Client in the registration phases for each domain. The Token Client then builds Token Request structures for each domain as described later in this section.

After building the Token Request structures, the Token Client broadcasts them, either using traditional broadcasts techniques or using predefined destinations of Pervasive Authentication Domains that can be set at registration time. If no response is received, the Token Client tries additional broadcasts until it receives a response or gives up, in which case a failure message is returned to the client application. Once a Token Response is received, the response is checked for integrity and validity and the nonce is compared to the nonce sent in the Token Request. If the response is valid, the credential is extracted and given to the client application, which can then access the external service on behalf of

the user.

The Token Server listens for Token Requests from Token Clients, and checks incoming Token Requests for integrity and validity by decrypting and comparing SlaveID and nonce fields. The Token Server retrieves the SlaveID and the DomainID of the pervasive device which is included in the Token Requests, and determines if the device is permitted to receive tokens by consulting with the Policy component and by determining if the user is logged into the domain. The Policy component allows tokens to be released if DomainID in the Token Request matches the domain identification of the Token Server, if the SlaveID obtained from the Token Request has been registered with the gateway for the given DomainID, and if the pervasive device is deemed to be authorized by the Personal Authentication Gateway as described in Section 2.3. If tokens can be sent, they are fetched by the Token Server from the Token Store, and a Token Response is built and sent. If tokens cannot be sent, a rejection message is sent to the Token Client.

Message Format and Protection. A Token Request structure, illustrated in Figure 3a contains the DomainID and corresponding SlaveID for the pervasive devices in the domain. The SlaveID provides additional logical addressing for several Automatic Token Clients sharing the same physical addressing. The DomainID distinguishes different domains, such as a home domain or an office domain.

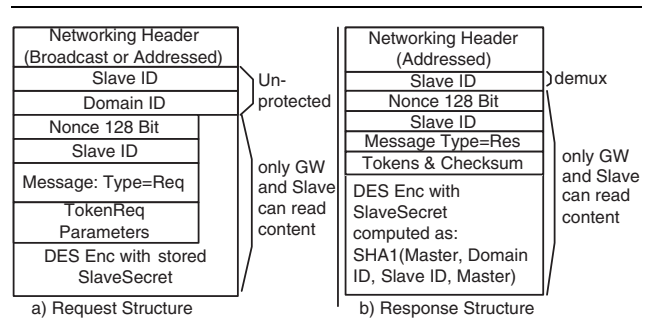


Figure 3. Message Formats

A 128-bit nonce field, Nonce128bit, is randomly generated by the pervasive device and included in the Token Request to help protect against Token Request replay attacks. A Message Type identifier which describes the type of the message (e.g., a Token Request), the token request parameters which include information needed to obtain the token (integrity checksums, the challenge from a challenge-response, etc.), the SlaveID, and the nonce are encrypted using a symmetric cryptographic encryption algorithm (e.g., Triple-DES in CBC mode) with the result of the encryption also included in the Token Request. The DES encryption

key is the SlaveSecret corresponding to the SlaveID which is obtained securely at registration time as a secure hash (e.g., SHA1) of the SlaveID, DomainID and a master key known to the gateway. The DES encryption allows the Personal Authentication Gateway to be sure that the Token Request is from the Pervasive Device to which the SlaveSecret is known. The nonce field is used in the Token Response so that the Token Client can verify that the Token Response is valid. Because the nonce field is encrypted by secrets known only to the pervasive device and the Personal Authentication Gateway, receiving a Token Response with a nonce repeating a nonce from a Token Request implies an association between the Token Request and Token Response. The encryption ensures that only the Personal Authentication Gateway can read the nonce and the message type, and it provides an integrity-protected copy of the SlaveID.

A Token Request is validated at the Personal Authentication Gateway by first checking the Policy component to see if the Token Server is authorized to distribute tokens to a Token Client with the SlaveID and DomainID listed in the Token Request. If so, the Token Server decrypts the DES-encrypted portion of the Token Request using the SlaveSecret DES key established at registration time revealing the decrypted Nonce128bit field, SlaveID field, and Message Type field. The SlaveSecret can be stored by the Token Server, or recomputed on the fly from the master key, DomainID, and SlaveID. The Token Server verifies that the decrypted SlaveID matches the unprotected SlaveID, and that the Message Type field indicates a Token Request. If the Token Server has tokens for SlaveID and DomainID in the Token Store, then the Token Request is valid.

A Token Response structure is illustrated in Figure 3b. The Token Response includes the SlaveID of the Token Client that issued the corresponding Token Request. It also includes the message type in the Message Type field (e.g., Token Response), and a Tokens&Checksum field which contains the authentication tokens and checksums for integrity. The Nonce128bit field containing the nonce from the Token Request, the SlaveID, the Message Type, and the Tokens&Checksum field are encrypted by the Token Server with triple-DES encryption keyed with the SlaveSecret established at registration. The DES encryption allows the Pervasive Authentication Gateway to ensure that unprotected and encrypted SlaveID fields match the pervasive device that can decrypt the tokens and protects the Tokens and Checksum from disclosure against other devices. Since the pervasive devices could be relatively insecure compared to the Personal Authentication Gateway, the tokens are protected with cookie-like [8] mechanisms to control which token to send to which service, the token validity period, and the token persistence (e.g. erased at reboot).

A Token Response is validated by the Token Client by first checking that the SlaveID field in the Token Client

matches a SlaveID of the Token Client. If so, the Token Client decrypts the DES-encrypted portion of the Token Response using the SlaveSecret DES key it obtained when the pervasive device registered with the Personal Authentication Gateway. The decrypted and the unprotected SlaveID fields are compared and if they match, the decrypted Message Type field is checked to see that this is a Token Response message. Next, the Token Client verifies that the decrypted nonce field matches the nonce it sent in a corresponding Token Request, and the Token Client computes checksums for the tokens returned in the Tokens&Checksum field, comparing them with the checksums in the field. If all of these checks succeed, the Token Response is valid.

2.3. Security Model

The security model describes boundary conditions required by the system to meet security goals. With access control systems as ours, a general goal is that access should be granted only to such devices that are authorized by their user. Hence, we proceed by describing what authorizes a device to represent its user when accessing remote services or responding to remote service requests.

Authorization Domain Model. A pervasive device is authorized to represent a user, if it resides in the authorization domain of its user. The authorization domain of a user is represented by the intersection of three domains: (1) the configuration domain of a user, being spanned by all devices that are configured through a registration procedure to represent a user, i.e., share a secret with the authentication gateway; (2) the environmental domain of a user, being defined by a distance requirement based on a distance metric around the user's physical location; the distance could be derived by the gateway by measuring the signal strength of wireless communications or by using GPS [10]; and (3) the integrity domain of a user, being defined by acceptable integrity measures for a device, e.g., supported by TCG-based [1] hardware and attestation of a pervasive device's integrity status.

Whether an authorized device can represent a user depends also on the availability of the Personal Authentication Gateway that enables a device by providing tokens to exercise its authority. In particular, credentials held by pervasive devices expire after a short time. Therefore long-term credential revocation is not necessary for pervasive devices that are distrusted, misplaced, or stolen. While the Personal Authentication Gateway offers a single point of attack, at the same time it is much better equipped to withstand attacks than the pervasive devices. A successful gateway attack will require the user to reset all long-term credentials that may have been disclosed by the gateway. Finally, a foreign pervasive device gaining knowledge of a valid slave secret will be able to obtain short-term credentials from the gateway

as long as it is within the physical boundaries of the domain (e.g. close enough based on wireless signal strength). Once TPM becomes available for pervasive devices, this attack can be countered by using secure TPM identities. If a slave secret is suspected to be compromised, a user can reset the pervasive device by deactivating the old SlaveID and registering a new SlaveID, thereby receiving a new slave secret; no long-term credentials or passwords need to be reset.

3. Related Work

Existing solutions offer some of the desirable qualities of Pervasive Authentication Domains. For example, user credentials can be protected if a device is lost or stolen as long as the user credential has limited time validity, or is not cached by the device. However, this implies that the user would need to frequently provide credentials to the device.

There are many solutions for exchanging data from one device to another that could be used to share credentials between pervasive devices. Such solutions include TCP/IP over wireless or PDA infrared hot-syncing protocols, among others. These solutions, however, do not securely authenticate the devices. In PDA hot-syncing for example, the only authentication used is the name of the device, which can easily be determined and forged.

The Security Assertion Markup Language (SAML [11]) is an XML-based [13] security standard for exchanging authentication and authorization information that is likely to be used to encode information that is exchanged between servers and merchants or clients. SAML offers one possibility for encoding information exchanged between the Personal Authentication Gateway and pervasive devices.

Kerberos [7] allows client applications the ability to have automated, repeated authentication to a particular service through the use of the Kerberos ticket. Our architecture makes use of Kerberos and allows pervasive devices to store and use a Kerberos ticket while retaining the long-term client secret only in the Personal Authentication Gateway and the Kerberos server. See Itoi [6] for such a system where the long-term client secret is stored in a secure co-processor.

There are also systems like Dynamic Host Configuration Protocol (DHCP) [2, 3] in which one device provides information that another device needs to gain access to a network. In DHCP, a DHCP server provides an Internet Protocol address, an address for a network gateway, and addresses for Domain Name Service machines to a DHCP client computer. The DHCP client computer uses these addresses to gain access to the network such that the needed information does not need to be manually configured on the DHCP client. However, the DHCP system does not address distribution of user credentials and cannot protect against disclosure of the information it provides to the client.

4. Conclusions

We have presented a security architecture for pervasive environments that allows multiple pervasive devices to participate in a Pervasive Authentication Domain, enabling them to represent the domain user. We use short-term credentials for pervasive devices, keeping long-term secrets manageable and secure inside the gateway. We enable computationally weak pervasive devices to represent the user by distributing short-term credentials to them with a computationally lightweight protocol. By sharing credentials automatically among multiple pervasive devices, we eliminate the static relationship between services and devices, allowing services to be distributed to devices on demand.

As a future enhancement, TCG-based attestation [1] could enable the Personal Authentication Gateway to verify a pervasive device's integrity state before distributing security configuration that allows it to represent the user. Another enhancement would be to integrate the configuration and token requests into the DHCP protocol. In this case, the same device would be used as a DHCP server and a Personal Authentication Gateway for configuring both networking and security of the pervasive devices.

References

- [1] Trusted Computing Group – Background, May 2003. https://www.trustedcomputinggroup.org/downloads/TCG_Backgrounder.pdf.
- [2] R. Droms. Dynamic host configuration protocol, March 1997. IETF RFC 2131.
- [3] R. Droms, J. Bound, B. Volz, T. Lemon, C. Perkins, and M. Carney. Dynamic host configuration protocol for ipv6 (dhcpv6), July 2003. IETF RFC 3315.
- [4] D. Eastlake and P. Jones. Secure hash algorithm 1 (SHA1), September 2001.
- [5] C. N. et. al. IBM's Linux Watch: The challenge of miniaturization. *IEEE Computer*, 35(1):33–41, January 2002.
- [6] N. Itoi. Secure Coprocessor Integration with Kerberos V5. In *9th USENIX Security Symposium*. USENIX, 2000.
- [7] J. Kohl and C. Neuman. Kerberos network authentication service (V5), September 1993. IETF RFC 1510.
- [8] D. Kristol and L. Montulli. HTTP state management mechanism, February 1997. Request for Comment 2109, Network Working Group.
- [9] D. Marculescu, R. Marculescu, S. Park, and S. Jayaraman. Ready to Ware. *IEEE Spectrum*, October 2003.
- [10] NAVSTAR Global Positioning System Joint Program Office. GPS technical library. <https://gps.losangeles.af.mil/gp-sarchives/1000-public/1300-lib/default.html>.
- [11] OASIS. Security assertion markup language (SAML), 2002. <http://www.oasis-open.org/committees/security>.
- [12] W. Simpson. PPP challenge handshake authentication protocol (CHAP), August 1996. IETF RFC 1994.
- [13] World Wide Web Consortium (W3C). Extensible markup language XML, 2002. <http://www.w3.org/XML>.