WWPass technology in depth

October 1, 2018

Table of Contents

Overview

WWPass may be considered as a Third-party Identity Provider. Other examples of Third-Party Identity providers include OpenID, SAML, CAS. OAuth in some aspects may also be considered as a Third-Party Identity provider.

Third Party Identity provider stores (personal) user data and provides access to the data to external Service Providers (Relying parties)

WWPass differs in the following main aspects:

- *it heavily relies on hardware crypto token*
- *there is no single set of user data, known to Identity Provider and distributed among Service Providers; instead every pair user/Service Provider has its own fully isolated data container*
- *there is no restriction on user data format - particular serialization as well as data fields is a choice and responsibility of Service Provider, the owner of these user data. WWPass may be considered as a per-user storage utility for Service Providers*

Another important features of WWPass is token management:

- *issuance and revocation of crypto tokens may be done by user alone, without WWPass active involvement*

WWPass transaction steps

Every user account (keyset) is assigned unique User Identifier (UserID), 128 bit long random number. UserID is kept in a token memory and may be read only by WWPass core network
Every Service Provider is assigned unique ServiceProvider Identifier (128 bit long random number), SpID

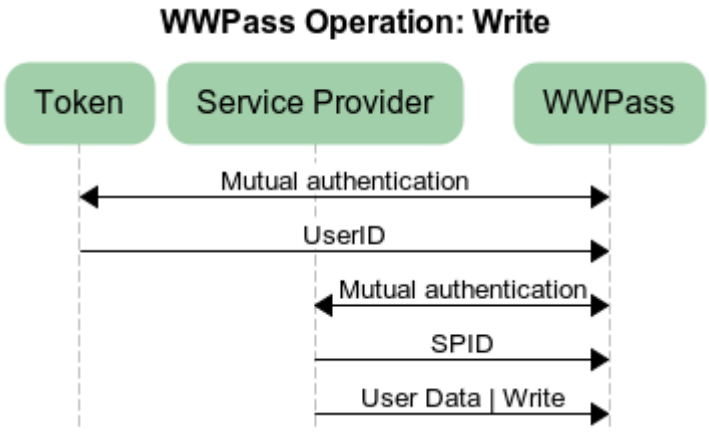
1. *User hardware crypto token and WWPass are mutually authenticated, UserID becomes known to WWPass*

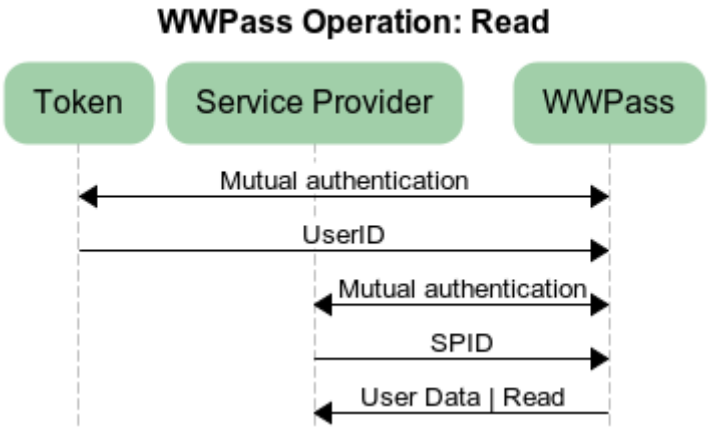
2. *Service provider and WWPass are mutually authenticated, SpID becomes known to WWPass*

3. *UserID and SpID pair is used to calculate address of data container*

4. *Read/Write access*

4.1. The content of this data container is sent form WWPass storage to Service Provider (read operation)
4.2. Alternatively Service Provider can send user data to selected WWPass data container (write operation)

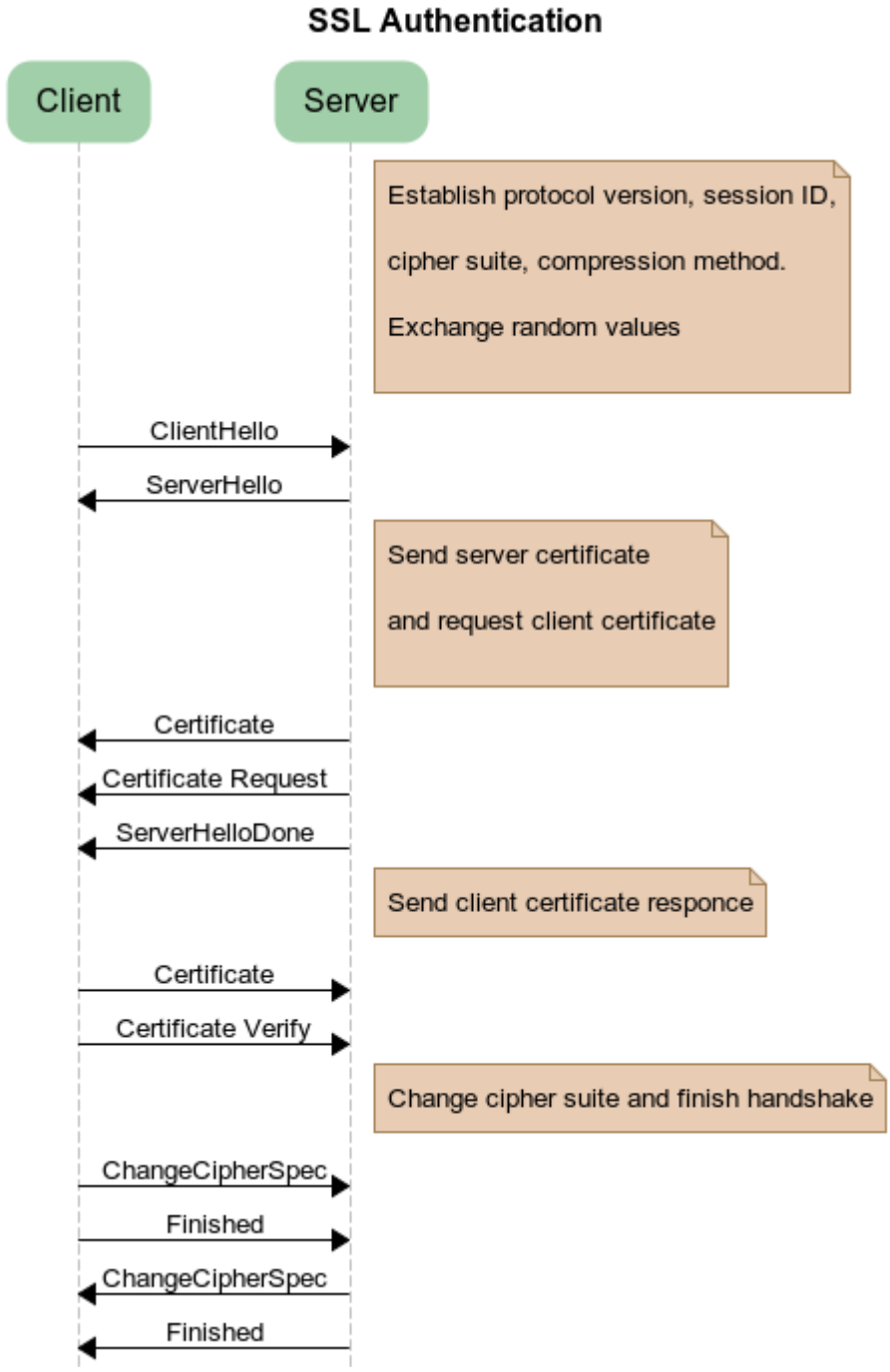




Authentication details: Service Provider / WWPass

Service Provider and WWPass mutual authentication uses SSL protocol with both server-side and client side certificates. Resulting SSL-connection is encrypted, further data transfer is secured.

To be able to participate WWPass transactions Service Providers should be registered at WWPass network. During registration Service Provider is assigned new unique Service Provider identifier (SpID) and unique ServiceProvider human readable name. The name is usually (but not necessary) based on SP URL. Finally, this registered Service Provider receives X509 certificate signed by WWPass CA.


SSL Authentication
Client
Server
Establish protocol version, session ID,
cipher suite, compression method.
Exchange random values
ClientHello
ServerHello
Send server certificate
and request client certificate
Certificate
Certificate Request
ServerHelloDone
Send client certificate responce
Certificate
Certificate Verify
Change cipher suite and finish handshake
ChangeCipherSpec
Finished
ChangeCipherSpec
Finished

Authentication details: User

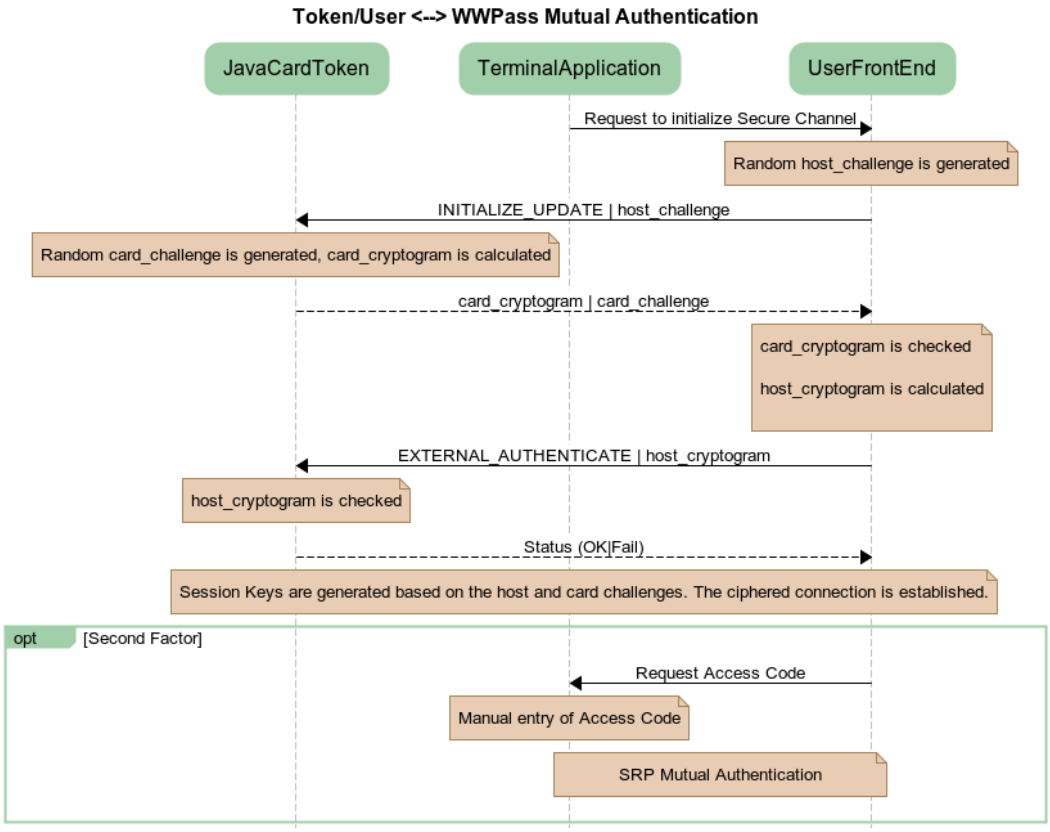
Token / WWPass

Token / WWPass mutual authentication is implemented according to ***GlobalPlatform Secure Channel protocol 03****. See Secure Channel Protocol 03 GlobalPlatform Card Specification v2.2 - Amendment D - 1.0 The protocol is based on a preshared secret key. As a result of mutual authentication new session key is generated which protects further data exchange.*

See also WWPass document "Appendix A. Java Card Token.pdf"

Access code (or PIN, or Password)

Hardware token authentication is WWPass mandatory first factor (something user has). Service Provider may request two-factor authentication. This time successful token authentication is followed by manual entry of "Access code" (something user knows). WWPass implements Secure Remote Password (SRP) protocol. See e.g. SRP Official site and RFC-2945. In the context of WWPass authentication SRP has its own peculiarities. SRP starts after token authentication. Hence the user account is already identified; "username" becomes redundant and a predefined constant string is used in calculations.



Data Container access

Data container storage is designed to be secure in many aspects.

- *Data is encrypted at rest. Encryption key may be only calculated in presence of user token*
- *Data Container identifier (address) cannot be used to identify user or Service Provider whom this Data Container belongs*
- *Data Containers are stored geographically dispersed*

Data Container address calculation

As a result of Token and Service Provider authentication WWPass gets UserID and SpID. Data container address is one-way function of (USerID | SpID) concatenation. To calculate data container address, WWPass uses asymmetric encryption with a predefined public key. The key was created for this particular purpose as a part of

public/private key generation. This public key is a lifetime constant of WWPass service. Private key of the pair is not used at all and may be safely destroyed.

Alternatively Zp encryption (raising p to the power of (UserID|SpID) concatenation) may be used for one-way address calculation

Data Container encryption

No matter if the Data Container is encrypted by Service Provider or not, it is always additionally encrypted by WWPass before sending data to the storage subsystem. WWPass uses another public key belonging to another key pair to encrypt (UserID | SpID) concatenation. Last 128 bits of the result are used as an AES key to encrypt the data. When reading data back, same calculations allow to restore encryption key. It is important to note that data at rest are thus encrypted. The encryption key is not stored in WWPass on a permanent basis and is only available as a result of Service Provider and token authentication.

Alternatively this Data Container encryption key may be calculated as a hash of (UserID | SpID) concatenation.

Dispersed data storage

WWPass keeps Data Containers in the 12-nodes Dispersed data storage (Reed-Solomon 6-out-of-12). This way WWPass can survive in case of loss or compromising of less then 6 storage modes. It is important that data is geographical distributed (all around the Globe)

Alternatively in particular implementation data distribution may be limited by country/region borders

Alternatively 6-out-of-12 Reed-Solomon distribution may be reduced to e.g. 2-out-of-3 nodes

Ticket as a transaction identifier

Mutual authentications in pairs (user ⇔ WWPass) and (Service Provider ⇔ WWPass) are not enough to address a data container with user data to the Service Provider. Since the communication between a Service Provider and a user is a many-to-many relationship, WWPass needs to identify a particular session between the user and the Service Provider. Tickets help to solve the task.

What is a Ticket

The term itself was borrowed from the Kerberos protocol, but the underlying mechanism is quite different. A Ticket is a small piece of information, circulating between a user, a Service Provider and WWPass. Its formal representation is:

<spname:>[p:]<nonce>@<issuer>

where

- <spname> is the registered human-readable name of the Service Provider; it is presented to the user in the consent dialog
- p is optional, when present indicates two-factor authentication (access code required)
- <nonce> stands for "number used once", random sequence of hex digits
- <issuer> is a WWPass network node where the ticket was created

Example:

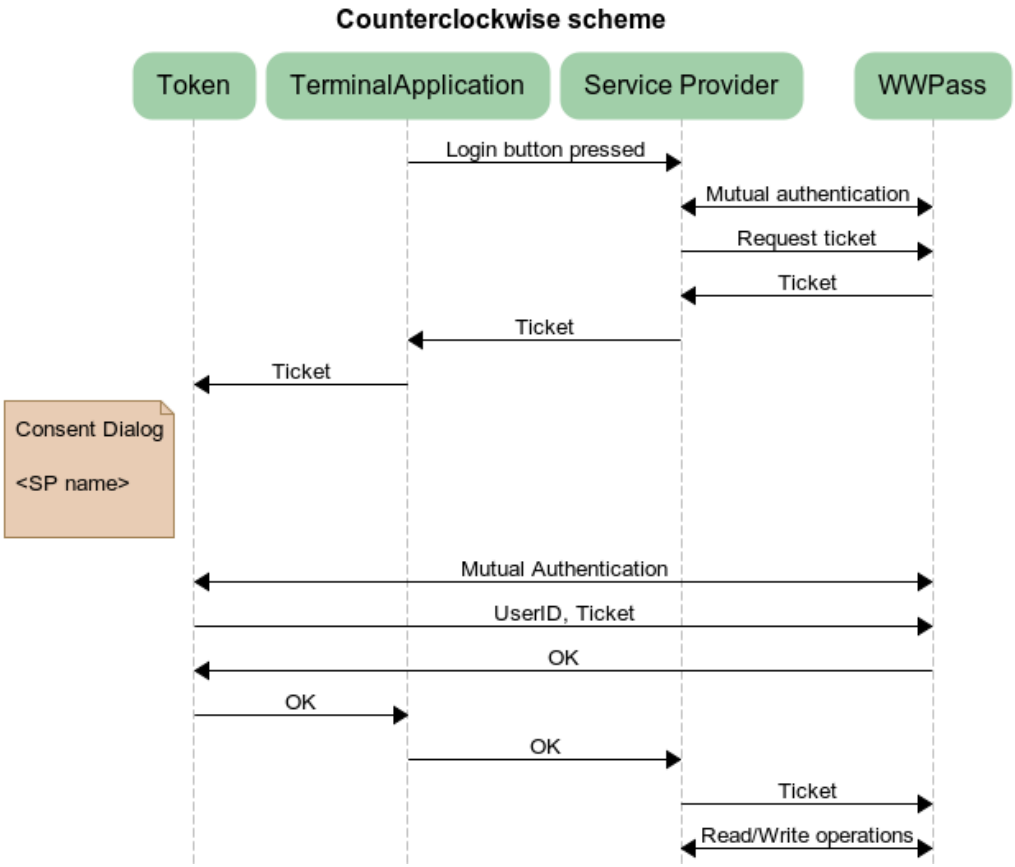
example.com:0be4a4c407f021a6e39032755fec65611ed6c758e7ead6309412c0cbb966fa08bd887c421b3d@WWPass_srv15:16032

Two slightly different schemes are used - "Clockwise" and "Counter-Clockwise".

Counter-Clockwise scheme

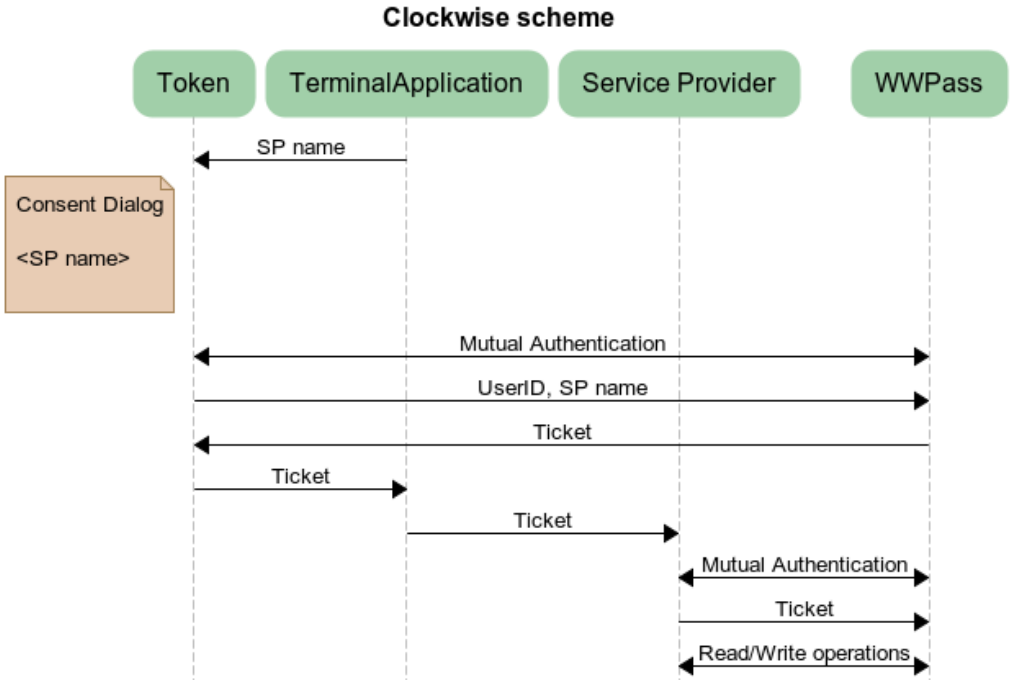
The user starts authentication by pressing a button on a client application (e.g. a web page in a browser). Service Provider connects to WWPass, authenticates itself and requests a ticket. WWPass generates new ticket with Service Provider name. WWPass also remembers an association between the ticket and Service Provider Identifier (SpID). Now Service Provider passes the ticket to the client application. The application in turn downloads the ticket into the token. Depending on the token ability to display Service Provider name (e.g. token on a smart phone) the "Consent dialog" is shown on token screen or on terminal display.

On user confirmation token authenticates at WWPass, establishes ciphered channel and sends UserID and the ticket. Now WWPass has can address particular DataContainer and provides access to it to Service Provider. The ticket is user as a transaction identifier between Service Provider and WWPass in future data exchange.



Clockwise scheme

The user starts authentication by pressing a button on a client application (e.g. a web page in a browser). Service Provider name is downloaded into the token and user is presented consent dialog (Allow authentication into - Yes/no). If user confirms, token authenticates to WWPass and sends SP name and UserID. WWPass generates a ticket with received SP name and remembers the association (ticket - UserID). Ticket is returned back to the token. Now tokens passes this new ticket to the Terminal Application, where from the ticket is received by Service Provider. Service Provider authenticates at WWPass and presents the ticket. WWPass checks if the SP name in the ticket belongs to the particular Service Provider and allows read/write access to corresponding Data Container.



Token management. Token life-cycle

Every user account (keyset) may include a number of tokens with the same UserID. Each of them provides access to the same data, defined by UserID.

Those tokens constitute a "keyset". User can deactivate any token from this set as well as to add new token.

To allow mutual authentication to WWPass each token has two life-time constants, TokenID and TokenSecret. The TokenID is unique throughout WWPass network and is used as "Key diversification data" according to GlobalPlatform terminology. TokenSecret is preshared secret used in mutual authentication according GlobalPlatform Authentication Protocol 03.

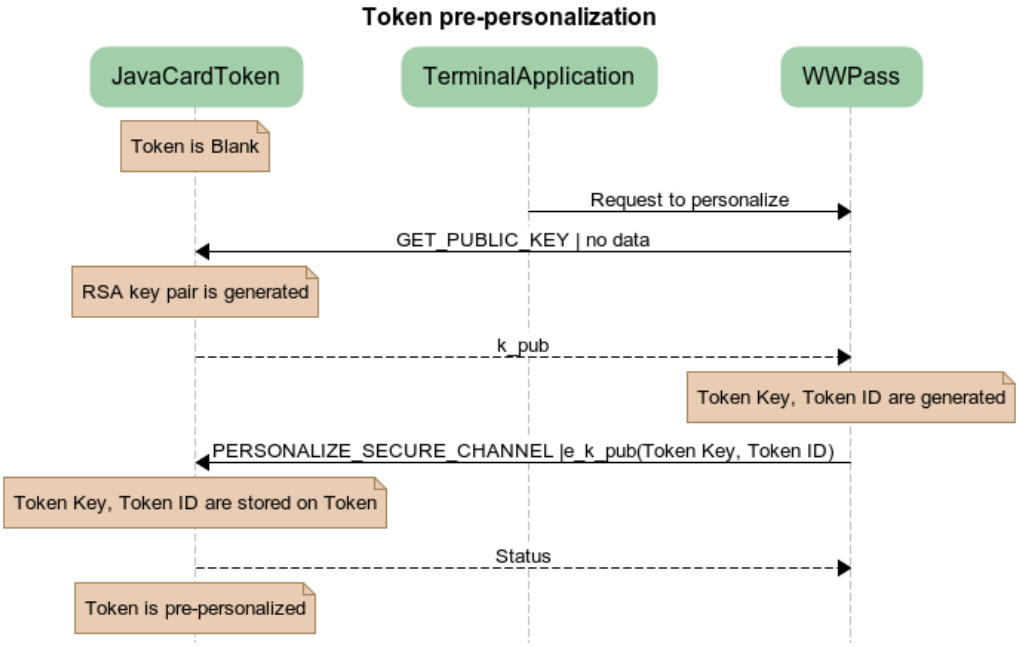
Every token has two other parameters, which are common for the keyset - UserID and UserDataEncryptionKey (See WWPass document "Appendix A. Java Card Token.pdf").

Blank token

WWPass cryptographic tokens are based on JavaCard microcomputers (see Java Card Platform Specification) During manufacturing a WWPass application (cardlet) is downloaded into the JavaCard. At this point it is a "blank WWPass token"

Pre-personalization

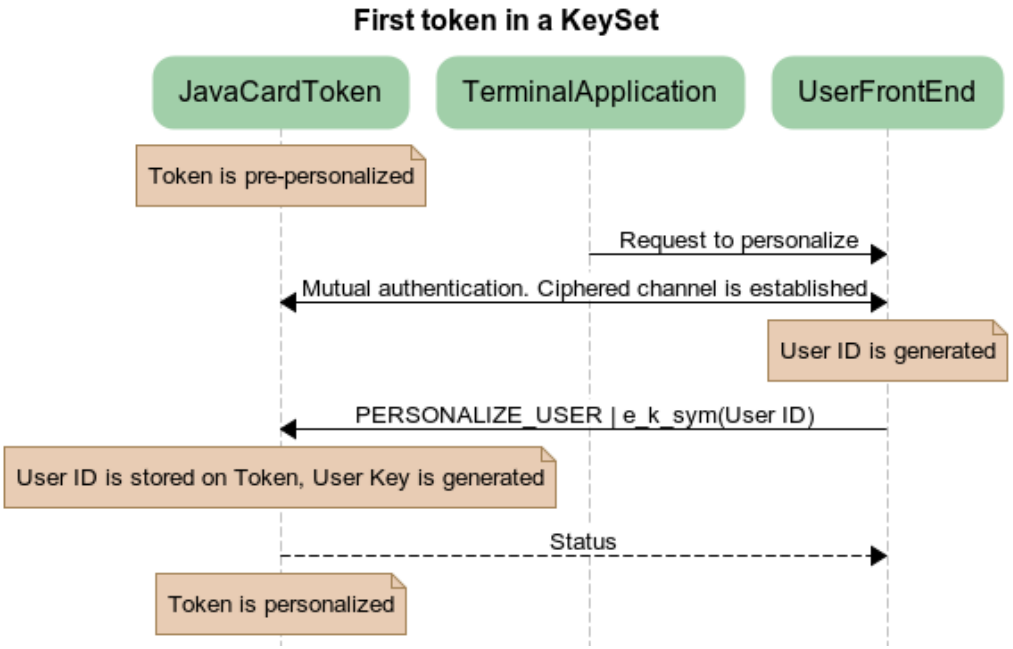
Next step is "pre-personalization". Blank token is connected to WWPass network, which allocates and downloads TokenID and TokenSecret into the device. Tokens are pre-personalized in a secure "factory" environment.



Personalization

Tokens with UserID and UserDataEncryptionkey are "personalized". UserDataEncryptionKey is used by some applications for cryptographic operations on user Terminal.

Pre-personalized tokens are delivered to end-users. Further events depend on whether it is the first token in a keyset or the token will be added to already existing keyset. First case - when activating a new keyset, pre-personalized token connects to the WWPass network via user terminal. WWPass allocates new unique UserID and sets corresponding constant in the token memory. Token generates a random 128 bit number - UserDataEncryptionkey. Its value is not known to WWPass network.



Adding a token to already existing keyset involves one already personalized token and another token in pre-personalized state. The procedure is coordinated by WWPass network.

First personalized token is connected to WWPass via user terminal. After mutual authentication the token:

- *WWPass generates random symmetric encryption key and sends it to the token*
- *token encrypts (UserID, UserDataEncryptionKey) bundle with the received key*
- *token outputs the result (bundle) to the UserTerminal memory for a temporary storage.*

Now user is advised to disconnect personalized token and connect new token (which is in prepersonalized state);

After mutual authentication at WWPass network the token

- *downloads the encrypted (UserID, UserDataEncryptionKey) bundle from User Terminal memory*
- *gets (via encrypted channel) the symmetric key, kept in WWPass*

- *decrypts the (UserID,UserDataEncryptionKey) pair and stores values in its memory*