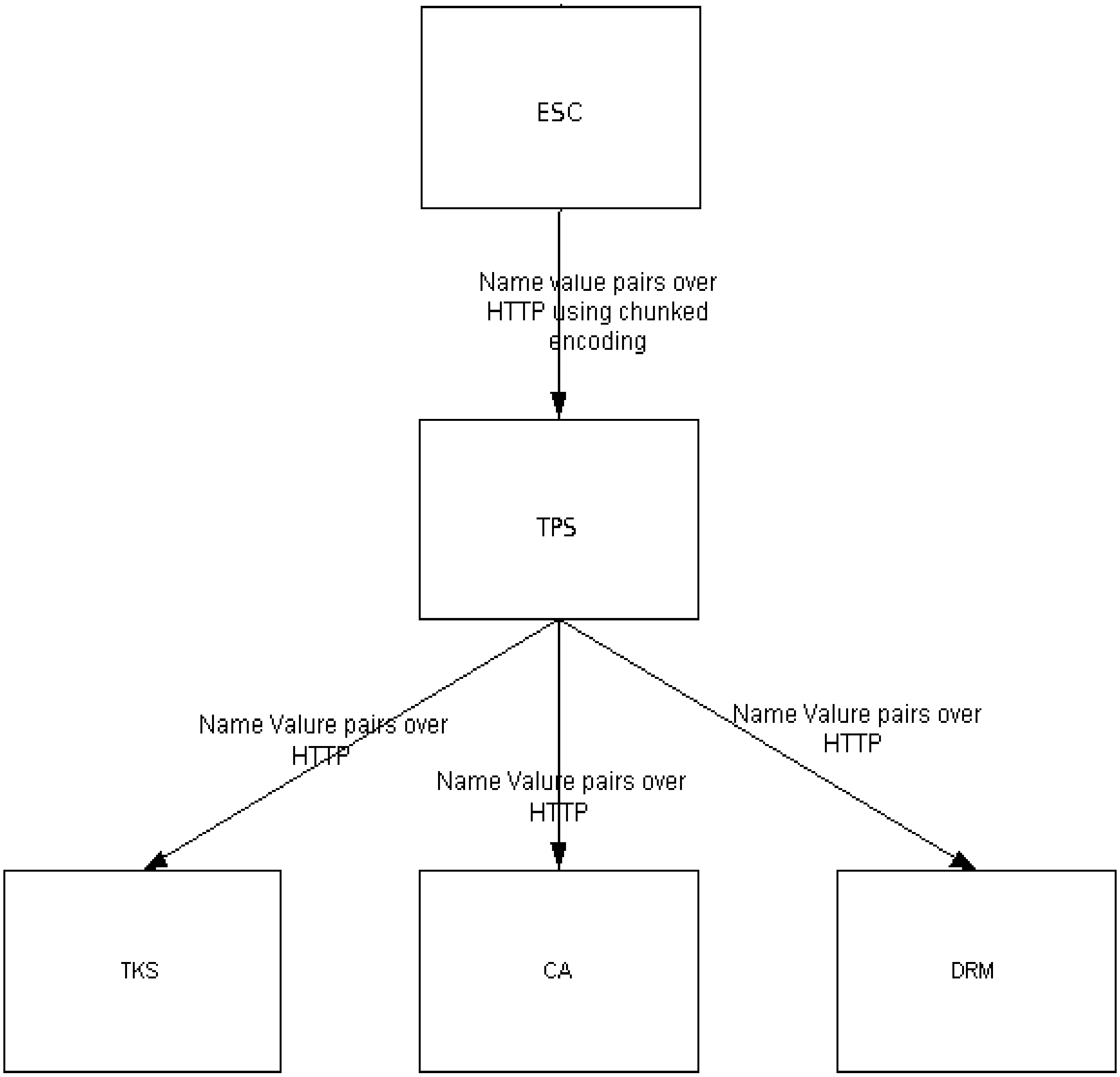
Protocol Summary between TPS Backend Components

2004/1/09 (Updated 2009/9/29)
Note: This document is informational only. Numerous changes have been made to the underlying source code since this document was originally written. Although every effort has been taken to ensure that changes have been reflected in this updated version some discrepancies may exist. Please refer to the source code and headers in the code repository for authoritative information.

1.0 Overview

The following picture shows various Token Management front-end and backend components.


ESC
Name value pairs over HTTP using chunked encoding
TPS
Name Value pairs over HTTP
Name Value pairs over HTTP
Name Value pairs over HTTP
TKS
CA
DRM

Components	**Description**
TPS	This component is responsible for

	• Format Operation • Handling Pin Reset Operation • Handling Enrollment Operation
CA	This component is responsible for • Issuing Certificates.
TKS	This component is responsible for • the generation of MACing session key and host cryptogram • encrypt data using one of the keys in the token key set • updating key sets for tokens.
DRM	This component is responsible for • the generation and archival of the user's encryption keys.

2.0 Protocols

This section details the protocols among back end components.

2.1 Protocol Between ESC and TPS

TPS is exposes its service in the url below:
The angle brackets "<>" are for the purpose of emphasizing the format here and are not part of the syntax.

```
http://<ra_host>:<ra_port>/nk_service
```

Each request and response is encapsulated as one chunk in HTTP1.1's chunked encoding.

```
s=<message_size>&msg_type=<message_type>&<parameters>

where,

The angle brackets "<>" are for the purpose of emphasizing the format here and are not part of the syntax
<message_size> should be the size of the message in bytes excluding 's=<message_size>&' portion
<message_type> is the message type. See section 2.1.1 and 2.1.2 for supported values
<parameters> is a set of message type specific parameters. See section 2.1.1 and 2.1.2 for supported
parameter names and values. Parameters are URI encoded
```

For example, a client could send the following to represent a “begin op” operation which is used by ESC to kick off other fundamental operations such as Format and Pin Reset.

```
s=22&msg_type=2&operation=3

where,
msg_type=2 means BEGIN_OP
operation=3 means RESET_PIN operation.
```

2.1.1 TPS accepts the following messages:

Message Type	Activated	Parameter Names	Parameter Values	Description
2		operation, [extensions]	The value for 'operation' parameter should be either 1, 3, or 5. ENROLL,RESET_PIN, FORMAT Ex: msg='msg_type=2&operation=5&extensions=tokenType%3DuserKey%26clientVersion%3DESC+1%2E0%2E1%26tokenATR%3D3B9F9681B1FE591F078025A000000056573635303000010080%26statusUpdate%3Dtrue%26extendedLoginRequest%3Dtrue%26 The “extensions” are optional parameters giving TPS additional info.	Begin Op
4	only when auths.enable=true in the CS.cfg of TPS.	user_id (screen_name),password	The values for 'user_id', and 'password' parameters should be string. Ex: s=33&msg_type=4&screen_name=user1&password=1234	Login Response
17	only when auths.enable=true in the CS.cfg of TPS.	UID,PASSWORD	The values for “UID” and “password” parameters should be a string. Ex: s=33&msg_type=17&UID=user1&PASSWORD=1234	Extended Login Response
12		new_pin	The value for 'new_pin' parameter should be a string. Ex: s=33&msg_type=12&new_pin=1234	New Pin Response
10		pdu_size,pdu_data	The value for 'pdu_size' parameter should be number. The value for 'pdu_data' should be the string representation of the URL encoded data. Each unprintable byte of the pdu data is represented by a	Token PDU Response

			'%' followed by two characters. Ex: s=111&msg_type=10&pdu_size=47&pdu_data=%9F%7F%2A%04%00%00%00%00%104%01%01%03%00%04%2A%00%00%00%0C%EB%07%40%F4%5B%00%04%00%00%00%00%98%C1%06%00%E4%FF%FF%FF%2A%00%00%00%60%5B%90%00	
15		current_state	This message merely echoes back to the server the last value of “current_state”. Each message in our protocol must have a message in each direction. Ex: s=27&msg_type=15&current_state=2	Status Update Response

2.1.2 TPS returns the following responses:

Message Type	**Activated**	**Parameter Names**	**Parameter Values**	**Description**
3	only when auths.enable=true in the CS.cfg of the TPS.	invalid_pw, blocked	The values for both 'invalid_pw' or 'blocked' should be either 0 or 1. 0 represents false, 1 represents true. Both are usually 0 in practice. Ex: s=33&msg_type=3&invalid_pw=0&blocked=0	Login Request
16	only when auths.enable=true in the CS.cfg of the TPS auths parameters in TPS control the UI displayed by ESC. Auth.instance.0.ui.description.en=This authenticates user against LDAP directory. …. auth.instance.0.0.	invalid_login,blocked,title	This style used when using the pop up LDAP Authentication method. The values for both 'invalid_login' or 'blocked' should be either 0 or 1. 0 represents false, 1 represents true. Both are usually 0 in practice. The value of title is a simple protocol understood by ESC, used to construct a simple auth popup on the screen. Ex: s=338&msg_type=16&invalid_login=0&blocked=0&title=LDAP+Authentication&description=This+authenticates+user+against+the+LDAP+directory.&required_parameter0=id%3DUID%26name%3DLDAP+User+ID%26desc%3DLDAP+User+ID%26type%3Dstring%26option%3D&required_parameter1=id%3DPASSWORD%26name%3DLDAP+Password%26desc%3DLDAP+Password%26type%3Dpassword%26option%3D	Extended Login Request

	ui.title.en=LDAP Authenticaiton			
11		minimum_length, maximum_length	The values for both 'minimum_length' or 'maximum_length' should be number. Ex: s=73&msg_type=11&maximum_length=10&minimum_length=4	New Pin Request
9		pdu_size,pdu_data	The value for 'pdu_size' pameters should be number. The value for 'pdu_data' should be the string representation of the URL encoded data. Each unprintable byte of the pdu data is represented in Hex by a '%' followed by two characters. Ex: s=68&msg_type=9&pdu_size=12&pdu_data=%00%A4%04%00%07%62%76%01%FF%00%00%00	Token PDU Request
13		operation, result, message	The value for 'operation' parameter should be either 1, 3, or 5. The value for "result" parameter is either 0 for success or 1 for failure. The value for "message" parameter are defined in: pki/base/tps/src/include/processor/RA_Processor.h Ex: s=42&msg_type=13&operation=5&result=0&message=0	End Op
14		current_state,next_task_name	current_state is a number between 1-100 that gives the client an idea how far the process is along. next_task_name is a string representing the next task to take place in the process. Client currently makes no use of next_task_name. Ex: s=67&msg_type=14&current_state=10&next_task_name=PROGRESS_APPLET_UPGRADE'	Status Update Request

Here provided a sample transaction. (Work in progress)

From ESC to TPS	**From TPS to ESC**	**Description**

POST /nk_service HTTP/1.1 Host: broom:1924 Transfer-Encoding: chunked 1b s=22&msg_type=2&operation=3		Note that "Transfer-Encoding: chunked" is being used so that the server will response in chunked encoding. "1b" is the size of the chunk in hex representation.
	HTTP/1.1 200 OK Server: Netscape-Enterprise/6.1 Date: Thu, 09 Oct 2003 20:37:47 GMT Content-type: text/plain Transfer-Encoding: chunked 26 s=33&msg_type=3&invalid_pw=0&blocked=0	
...	...	
	4f s=74&msg_type=9&pdu_size=18&pdu_data=%84%04%...%DA%81	
2b s=38&msg_type=10&pdu_data=%90%00&pdu_size=2		
	0	0 as chunk size indicates that there is no more chunk and the client should terminate.

2.2 Protocol Between TPS and CA

The square brackets "[]" are for the purpose of emphasizing the format here and are not part of the syntax

From RA to CA	**Response From CA to RA**	**Description**

POST //ca/profileSubmitSSLClient?profileId=caTokenUserSigningKeyEnrollment&
tokscreenname=**[TOKEN_CUID]**&
publickey=**[PUBLIC_KEY]**
HTTP/1.1
		The profileId shows that this is
		for "House Key" enrollment.
		the **[TOKEN_CUID]** is the house key token's
		Card Unique Identifier, which is a verifiable unique value on the token.
		The **[PUBLIC_KEY]** is the normalized public key value.
		Each unprintable byte of the public key value is
		represented by a '%' followed by two characters.

	HTTP/1.1 200 OK^M Server: Netscape-Enterprise/6.1 ^M Date: Mon, 01 Dec 2003 23:25:22 GMT^M	Note, currently the response format is a Javascript inherited from browser enrollments. It may be streamlined later for real production.

	Content-type: text/html^M Content-length: 5036^M ^M **[response including certificate if successful]**	

POST //ca/profileSubmitSSLClient?
profileId=caTokenUserSigningKeyEnrollment&
screenname=**[USER_ID]**&
publickey=**[PUBLIC_KEY]**
HTTP/1.1
		The profileId shows that this is
		for "Net Key" enrollment.
		The **[USER_ID]** is an account user name.
		The **[PUBLIC_KEY]** is the normalized public key value

	HTTP/1.1 200 OK^M Server: Netscape-Enterprise/6.1 ^M Date: Mon, 01 Dec 2003 23:25:22 GMT^M Content-type: text/html^M Content-length: 5036^M ^M **[response including certificate if successful]**	Note, currently the response format is a Javascript inherited from browser enrollments. It may be streamlined later for real production.

Here provided one sample transaction for each profile (SigningKey and EncryptKey):

From TPS to CA	**Response From CA to TPS**	**Description**

POST //ca/profileSubmitSSLClient?
profileId=caTokenUserSigningKeyEnrollment&
tokencuid=0000305600001c3eff00&
publickey=MIGfMA0GCSqGSIb3DQEBAQU
AA4GNADCBiQKBgQCfLJSRHNh7v6k7cV%2Fix
FrDy2B4%0D%0AOuJB7Eejh25LRMTZpIFanEpZFG
23yBp0ZiQlWQp4L2mqE%2BIh2cfO9otzHv%2BajM0K
%0D%0AuPKh7HlYcuFxXJFiyYN0KVJEanRR%2FInGo
2wuespYB9lXChqVl6GoNmo%2FRGntEgzl%0D%0AhGs
GtoHxlYoFpsf0RwIDAQAB
 HTTP/1.1		

	HTTP/1.1 200 OK^M Server: Netscape-Enterprise/6.1 ^M Date: Mon, 01 Dec 2003 23:25:22 GMT^M Content-type: text/html^M Content-length: 5036^M ^M **[...response including certificate if successfu...l]**	Note, currently the response format is a Javascript inherited from browser enrollments. It may be streamlined later for real production.

POST //ca/profileSubmitSSLClient?
profileId=caTokenUserEncryptKeyEnrollment&
screenname=msg4cfu&
publickey=MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDM9uZ16%2BeyF9ki%2BA%2F3PZjQDuWA%0D%0A1NWg%2Fo%2Fg8aoU7xWniMwMUzc2aSQ%2F1kceD%2BVWiYX3D7YsUpI5Qw7ohGKDLYsCIhtD%0D%0AK1L18MYBUx1z4uDNU2uV8N26fSaGRlu0%2BNLNXGYUf4PDhPocQj07nVPWqFCWTSTU%0D%0AcCY8sUM1hMpfpbb93wIDAQAB
HTTP/1.1		

	HTTP/1.1 200 OK^M Server: Netscape-Enterprise/6.1 ^M Date: Mon, 01 Dec 2003 23:25:22 GMT^M Content-type: text/html^M Content-length: 5036^M ^M **[...response including certificate if successful...]**	Note, currently the response format is a Javascript inherited from browser enrollments. It may be streamlined later for real production.

2.3 Protocol Between TPS and TKS

TKS is the component that manages the master key(s), the transport key(s) and the token keys. The token keys may not be stored physically in TKS but they can be generated dynamically by using the CUID and the master key. Due to the importance of these keys, our requirement is that the token (Mac Key, Auth Key, and KEK Key) keys should never leave TKS. (We may need to adjust this requirement if we may to do key updates (diversification) from the server).

The channel between TPS and TKS is protected by SSL with client authentication. The client certificate must be registered in TKS as an agent certificate prior to any operations.

TKS supports the following requests:

Request	**URI**	**HTTP POST Parameters**	**Output Parameters**	**Remark**

ComputeSessionKey Request	https://<host>:<agent_port>/tks/computeSessionKey	CUID=<CUID>&
		card_challenge=<CardChallenge>&
		host_challenge=<HostChallenge>&
		KeyInfo=<KeyInfo>&
		card_cryptogram=<Card Cryptogram>
		
		where,
		
		<CUID> - Token ID
		<CardChallenge> - Card Challenge
			status=<status code>&
			sessionKey=<Session Key>&
			hostCryptogram=<HostCryptoGram>&
			encSessionKey=<Encryption Session Key>
			
			where,
			
			<status> - 0 for success, non-zero for failure.
			<Session Key> - Session Key that RA used to mac
				TKS is to calculate the "card cryptogram" and compare that with the card_cryptogram sent in by the token. The status will be non-zero if they don't match. TKS will also generate a "host cryptogram" to be sent back to the card for the card side of authentication. Retrieving the Mac'ing session key and

		<HostChallenge> - Host Challenge <KeyInfo> - Master Key ID (we could use Key Info Data from the token) <CardCryptogram> - value generated by the token that is to be verified by TKS as part of the authentication to complete session key agreement.	APDUs for net key token. <HostCryptoGram> - value generated by TKS that is to be verified by the token as part of the authentication to complete session key agreement. <Encryption Session Key> - Session Key to be used to encrypt APDU messages. This key is used when encryption is turned on.	encryption session key for RA.
EncryptData Request	https://<host>:<agent_port>/tks/encryptData	data=<Data>& CUID=<CUID>& KeyInfo=<KeyInfo> where, <Data> - Data to be encrypted with the KEK key in TKS (i.e. challenge) <CUID> - Token ID <KeyInfo> - Master Key ID (we could use Key Info Data from the token)	status=<status code>& encryptedData=<Encrypted Data> where, <status> - 0 for success, non-zero for failure. <Encrypted Data> - Encrypted Challenge [16 bytes]	This is for Proof of location. RA generates a challenge for enrollment, it sends plaintext challenge to TKS to encrypt, the encrypted challenge is sent to the NetKey token. The NetKey token will decrypt it with its KEK Key and put it in the proof-of-location which is part of the certificate request.. RA then verifies the proof.
KeySetChange Request	https://<host>:<agent_port>/tks/createKeySetData	newKeyInfo=<NewKeyInfo>& CUID=<CUID>& KeyInfo=<KeyInfo> where, <NewKeyInfo> - New Master Key Id (the most current Master key id created and used by TKS) <CUID> - Token ID <KeyInfo> - The Master Key Id used by the token note: Each byte of all binary data are represented by a two-character hex value preceded by a '#'	status=<status code>& keySetData=<newKeySet> where, <status> - 0 for success, non-zero for failure. <newKeySet> - key set generated with new master key. This key set is encrypted with the old KEK (key encryption key) from the old set for the token.	This is for key diversification which occurs when the master key has been replaced on TKS. RA has the most current master key version number. When detected, tokens with outdated master key version number will received a set of new keys generated based on the new master key. This function also works backwards to revert to older keys.

Here provided a sample transaction.

From TPS to TKS	**From TKS to TPS**	**Description**

POST /tks/computeSessionKey HTTP/1.1
Host: broom:1924
 	
CUID=#af#12#00#cc...&card_challenge=#21#3a.... &
host_challenge=#33#3b...&
KeyInfo=1&
card_cryptogram=#4c#3a...		

	HTTP/1.1 200 OK
	Server: Netscape-Enterprise/6.1
	Date: Thu, 09 Oct 2003 20:37:47 GMT
	Content-type: text/plain
	
	status=0&
	sessionKey=#f2#82#a2...&
	hostCryptogram=#11#22...&
	encSessionKey=#a1#81$43...	

2.3 Protocol Between TPS and DRM

TBD