SecureTV™
Technical Guide

Version 1.7

UpdateLogic Inc.
(508) 624-8688 (TEL)
(508) 624-8686 (FAX)
http://www.updatelogic.com

Thank you for using the UpdateLogic NetReady™ Service Management System. UpdateLogic provides comprehensive, cost-effective services which allow remote support sessions, software and firmware updates and secure field provisioning of DRM keys in all types of Internet enabled CE devices.

If you find any problems with this software, please report them via email to techsupport@updatelogic.com.

Please visit our website at www.updatelogic.com for the latest news and information.

UpdateLogic Incorporated

508-624-8688 (TEL)

508-624-8686 (FAX)

1 Confidentiality and Restricted Distribution

2 Change Tracking

Version Number of This Document	Changes
1.7	New platform integration section. Moved and reformatted CSP API. Removed security models section and replaced them with an expanded chain of trust section. Removed related documents section. Removed Terms Definition section. Added NetReady SMS front page. Updated graphics.
1.6	Renamed from NetProvision to SecureTV. New UtvProjectOnForceNOCContact() call described. Force argument removed from description of UtvProjectOnNetworkUp().
1.5	Added documentation for UtvProjectOnObjectRequestSize()
1.4	Updated security model section. Added peer stack section.
1.3	Minor fix and TOC update.
1.2	New overview diagram and new models of provisioned object protection section. New terms and related documents sections.
1.1	Errors corrected, better explanations, View Device Information NOC page added.
1.0	Original

3 Table Of Contents

4 SecureTV Overview

SecureTV is a facility that securely delivers streaming media authentication and encryption keys to Internet-connected appliances in a consumer's home. This facility is agnostic about key payload. Key payload may consist of SSL certificates, DRM authentication hash material, encryption keys, or e-commerce identity tokens. SecureTV can deliver any device-unique data to an Internet-connected appliance. This data is delivered initially when the device first connects to the Internet in the consumer's home and again as necessary when a key expires or needs to be

revoked and renewed due to compromise or upgrade. New keys for services and applications that are added after an appliance is shipped can be introduced as well.

SecureTV has two main pieces:

- The *Network Operating Center* (NOC) and its associated database and web interface. The NOC contains the key payloads that devices are provisioned with. A secure import facility is part of the NOC.
- The *Device Agent* (Agent) resides on the device. The Agent contacts the NOC to receive keys. CSP client programs that reside on the device contact the Agent to get access to the provisioned keys.

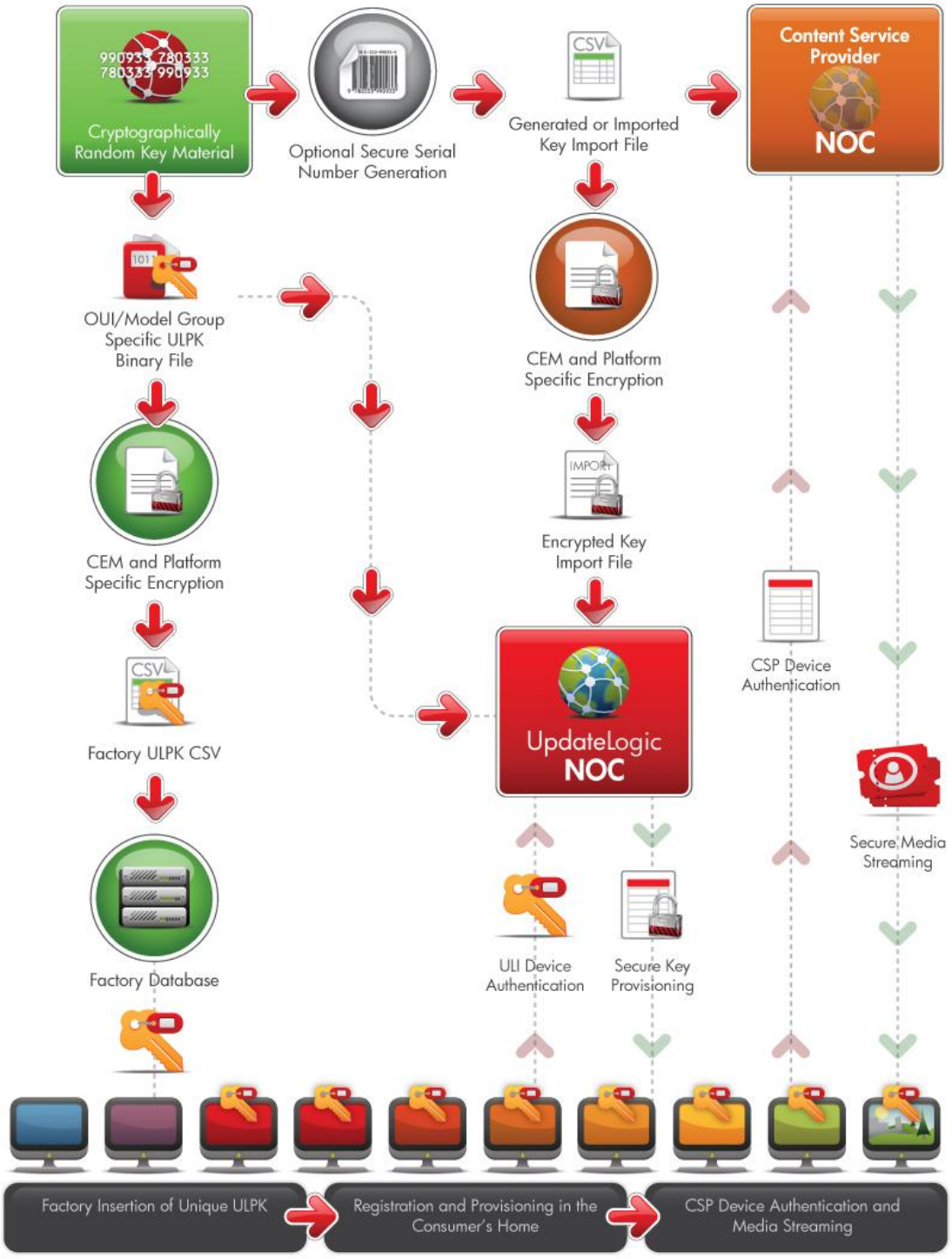
There are two categories of keys that SecureTV delivers.

- Device-unique keys that usually have a serial number associated with them for identification after provisioning. An X.509 client certificate and its associated private key are examples of device-unique keys. Device-unique keys are an example of many-to-many distribution. Device-unique keys are stored on the NOC and individually delivered to each device as it registers.
- Device-common keys that are common to entire class of devices. Device-common keys are an example of one-to-many distribution. One device-common key is stored on the NOC and is delivery to each device as it registers.

The following diagram illustrates the basic flow of SecureTV factory anchored, field delivered credentials.

SecureTV™

Industry-Approved Field-Delivered
Secure Credential Provisioning



5 Design Principles

5.1 Provisioned Object Opacity

SecureTV takes a Swiss bank account approach to the objects it provisions. Provisioned objects are considered to be *opaque*. Their contents and operation do not need to be exposed to SecureTV. The only exception to this is when the DRM provider shares the contents of those objects so that ULI can generate and encrypt them. If the objects are not shared they will be encrypted as is. Objects are imported to the NOC only after they are encrypted. They are then securely delivered to the device when that device registers with the NOC. On the device the objects are delivered by an obfuscated name to a requesting authenticated client. At no point does SecureTV need to understand what they are used for or become involved in the DRM they support.

5.2 Chain of Trust

SecureTV's cryptographic chain of trust is always anchored in a device-unique SoC-specific hardware protected key called K-SOC-Unique. K-SOC-Unique is the root of trust key. The implementation of this key differs from one SoC to the next. K-SOC-Unique and the cipher that uses it is *not* shared with ULI.

A second hardware-protected key called K-SOC-Transit is a device-common, hardware protected key that is *pre-shared* with ULI along with the cipher that will use it on the device. The implementation of this key differs from one SoC to the next. At ULI, K-SOC-Transit is stored in a secure offline database and is used to encrypt UpdateLogic Provisioning Keys (ULPKs) so that they are secure while travelling from ULI until they're secured by K-SOC-Unique at the point when they're inserted into the device at the factory.

All devices have UpdateLogic Provisioning Keys (ULPKs) inserted into them at the factory. As stated above ULPKs are protected by K-SOC-Transit while in transit from ULI until they're inserted whereupon they must be protected by K-SOC-Unique before being stored to flash.

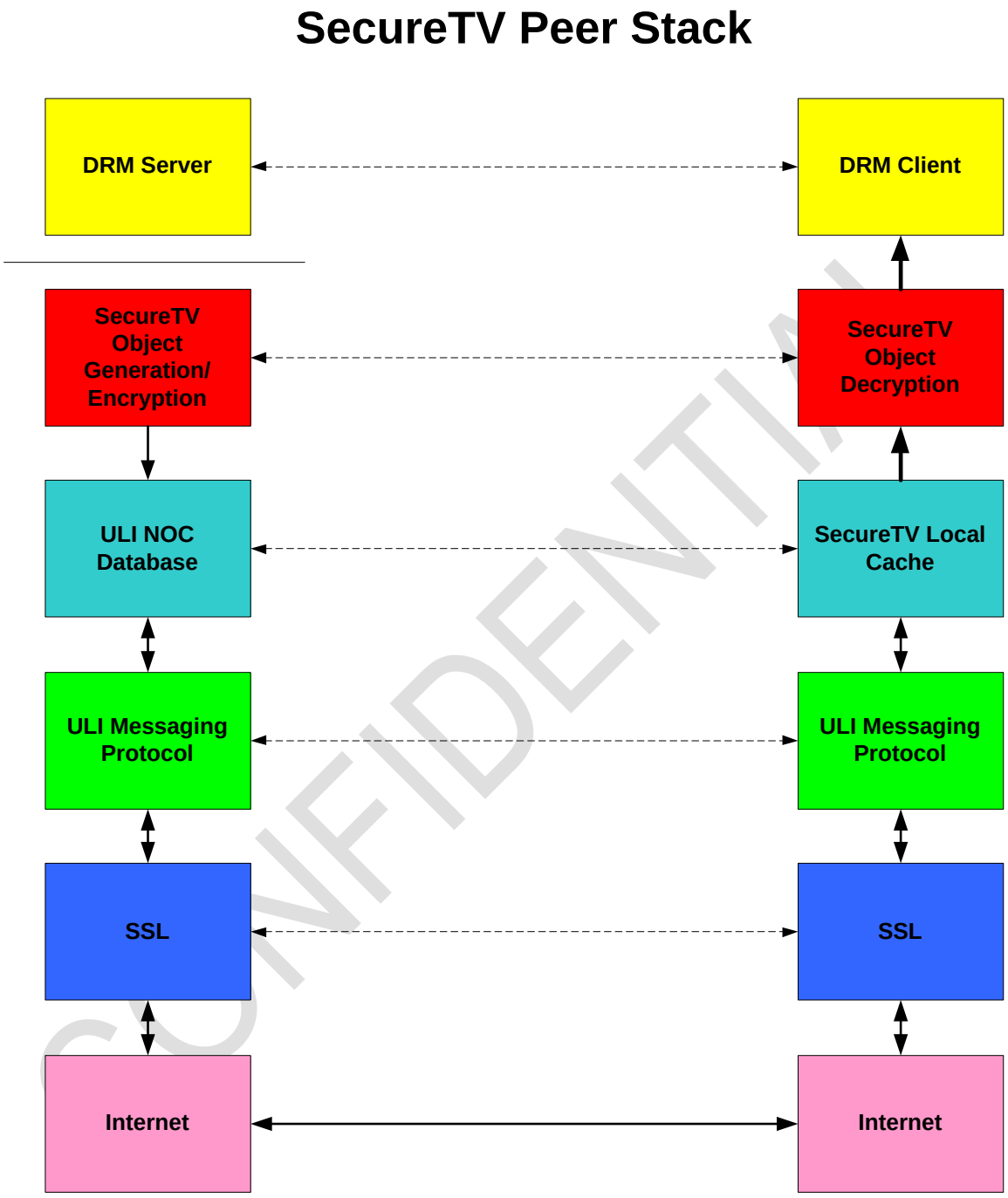
All provisioned objects are encrypted with a cryptographically random device–unique key when they are generated. This key is called K-SW-Unique and is stored in the ULPK. When a device registers with the ULI NOC it tells the NOC which ULPK was inserted into it. The NOC then provides the device with provisioned objects that were encrypted with the K-SW-Unique that corresponds to that ULPK. Provisioned objects are stored in the NOC in this encrypted form.

When a device connects to the internet for the first time provisioned objects are sent from the NOC to the device and stored in what is called the *local cache*. The local cache must be secured by K-SOC-Unique before it is written to flash.

Provisioned objects are decrypted before being returned to the requesting party.

5.3 Peer Stack

The following diagram depicts the peer relationships between SecureTV components on both the server and the device. At the top the DRM server converses with the DRM client on the device. The DRM client is being provided with security credentials that have arrived via SecureTV. These objects were generated or imported. The objects are being served from the SecureTV local cache which is a peer to the NOC database. The NOC and the device use the ULI messaging protocol to transport the provisioned objects. This transport occurs over the internet and is protected by an encrypted SSL session.



5.4 Key Generation and Import

SecureTV can either generate or import keys depending on the DRM being used. Some DRM protocols use the X.509 standard. These keys can be easily generated. Some DRM protocols use proprietary cryptographic key generation techniques that the DRM provider is willing to share so that ULI can generate them. Other DRM protocols use proprietary cryptographic key generation techniques that a DRM provider is not willing to share so ULI imports these keys from an

external source. All keys are encrypted so that only the target device can decrypt them and they are then stored in the UpdateLogic NOC database. All keys are given an obfuscated *key name* and *owner name* when they are imported. The names are pre-shared with the CSP's client program so that the client knows how to ask for their keys.

ULI engineers handle all key generation, encryption, and importing in coordination with CEMs, CSPs and DRM providers.

5.5 Registration and the ULPK

When a device first registers with the NOC in the consumer's home it uses an UpdateLogic Provisioning Key (ULPK) that was inserted at the factory to authenticate the device as valid to receive field provisioned keys. ULPKs are described below in detail. During registration and then approximately every 24 hours the Agent asks the NOC if any keys need to be provisioned to the device. The encrypted keys and their names are then sent to the device along a secure channel and stored on the device in their encrypted form.

5.6 The Local Cache

The key names together with the keys themselves make up what is called the *local cache*. The local cache is managed by the Agent. When a key retrieval request is initiated by a CSP's client program the Agent looks up the named key and returns it to the caller. The key may be decrypted before it's returned or it may be left encrypted for the caller to decrypt depending on the keys encryption type.

5.7 Revocation, Renewal, and Blacklisting

ULI engineering manages keys via a web interface to the NOC. They may be revoked and renewed. The Agent and NOC communicate so that what's done on the NOC is reflected in the device's local cache. Devices may also be blacklisted which causes all connected services including update to be denied. ULI only acts to deny service or revoke security credentials in cooperation with the CEM. These actions are never performed unilaterally.

6 SecureTV API

6.1 Device Agent Platform Adaptation Functions

The chain of trust described above is accomplished through the implementation of five (5) platform adaptation functions found in the Agent project layer. The integrator implements these functions so that they translate the general purpose cryptographic needs of the Agent into the SoC-specific calls that carry them out. This section describes those five calls. The prototypes for these functions can be found in utv-platform.h

6.1.1 UtvPlatformSecureOpen()

This function is called once when the Agent is first initialized using the UtvProjectOnBoot() call. (Please see the NetReady Agent Porting Instructions for more information about UtvProjectOnBoot().) The integrator should replace the body of this function with their SoC-specific security layer initialization code. This code could obtain an instance handle that would be stored in a static variable and used in the implementation of the functions that follow.

The prototype for this function is:

```
UTV_RESULT UtvPlatformSecureOpen( void )
```

It takes no arguments and must return UTV_OK if successful or a non-zero platform-specific error if there is a problem initializing the SoC's security layer.

6.1.2 UtvPlatformSecureClose()

The obvious complement to UtvPlatformSecureOpen(). Is called once when the Agent is shut down by the UtvProjectOnShutdown() call. (Please see the NetReady Agent Porting Instructions for more information about UtvProjectOnShutdown().) The integrator should replace the body of this function with their SoC-specific security layer close code.

The prototype for this function is:

```
UTV_RESULT UtvPlatformSecureClose( void )
```

It takes no arguments and must return UTV_OK if successful or a non-zero platform-specific error if there is a problem closing the SoC's security layer.

6.1.3 UtvPlatformSecureWrite()

UtvPlatformSecureWrite() takes a buffer, encrypts it using K-SOC-Unique and writes it to flash so that it's retrievable by the specified file name. It doesn't matter if the specified file name is preserved in the file system as long as when UtvPlatformSecureRead() is called using the same name the correct data is retrieved. This function is called at various times for various objects that the Agent needs to store securely. The integrator should replace the body of this function with SoC-specific security layer code to accomplish this. If the platform provides an encrypted file system that's based on K-SOC-Unique then the specified data can simply be written to that partition using fwrite().

The prototype for this function is:

```
UTV_RESULT UtvPlatformSecureWrite( UTV_INT8 *pszFileName,
UTV_BYTE *pubData, UTV_UINT32 uiSize )
```

The *pszFileName argument is a pointer to a string that contains a unique name for that object that may need additional platform-specific partition and path information pre-pended to it. *pubData is a pointer to the data. uiSize is the size of the data. The size of the objects will range from a about 16 bytes for the smallest object to somewhere around 50K bytes for the largest object. The largest object will be dependent on the size and number of provisioned objects delivered to the platform. The implementation of this function should round up the size of the data buffer provided as necessary to account for the padding required for the cipher employed by the SoC. The function must return UTV_OK if successful or a non-zero platform-specific error if there is a problem encrypting or writing the data.

6.1.4 UtvPlatformSecureRead()

UtvPlatformSecureRead() requests that an object with a specified name is read back from flash where it was placed by UtvPlatformSecureWrite(), decrypted, and placed in the provided buffer. This function will only be called for objects that have been previously stored with UtvPlatformSecureWrite(). If the platform provides an encrypted file system that's based on K-SOC-Unique then the specified data can simply be read back from that partition using fread().

The prototype for this function is:

```
UTV_RESULT UtvPlatformSecureRead( UTV_INT8 *pszFileName, UTV_BYTE
*pubBuff, UTV_UINT32 uiBufferSize, UTV_UINT32 *puiDataLen )
```

The *pszFileName argument is a pointer to a string that contains a unique name for the object to be read. It may need additional platform-specific partition and path information pre-pended to it depending on how UtvPlatformSecureWrite() was implemented. *pubBuff is a pointer to the buffer that is allocated to receive the data. uiBufferSize is the size of that buffer. *puiDataLen is the actual size of the retrieved data. The implementation of this function should remove any padding that was added by UtvPlatformSecureWrite() to maintain data fidelity. The function must return UTV_OK if successful or a non-zero platform-specific error if there is a problem decrypting or reading the data.

6.1.5 UtvPlatformSecureGetULPK()

UtvPlatformSecureGetULPK() requests that the ULPK be retrieved from wherever it was stored during factory insertion, and decrypted using K-SOC-Transit and K-SOC-Unique depending on how it was stored and encrypted during factory insertion. It's recommended that the ULPK is inserted with its K-SOC-Transit encryption intact into the secure file system that is protected by K-SOC-Unique. This would therefore require two decryptions to access the in-the-clear ULPK data which needs to be returned by this function. This function is called once when the Agent is first initialized using the UtvProjectOnBoot() call. (Please see the NetReady Agent Porting Instructions for more information about UtvProjectOnBoot().)

The prototype for this function is:

```
UTV_RESULT UtvPlatformSecureGetULPK(UTV_BYTE **pubULPK,
UTV_UINT32 *puiULPKLength )
```

**pubULPK is a pointer to a pointer where the pointer to the buffer containing the ULPK should be placed. The caller will call UtvFree() with that pointer when it's done with it. (Please see the NetReady Agent Porting Instructions for more information about UtvFree().) The *puiULPKLength argument is a pointer to a long where the size of the returned ULPK should be placed. The implementation of this function should remove any padding that was added during the ULPK insertion process. The function must return UTV_OK if successful or a non-zero platform-specific error if there is a problem decrypting or reading the ULPK.

6.2 Device Agent Streaming Client Interface Functions

This section describes the functions that a content streaming provider (CSP) client program uses to retrieve provisioned objects from the SecureTV Agent. The prototypes for these functions can be found in utv-core.h. Please note that unlike the platform adaptation functions described above these functions are system services that are already implemented.

6.2.1 UtvProjectOnObjectRequest()

UtvProjectOnObjectRequest() is used to retrieve a specified object into a specified buffer. The size of an object can be determined using the call UtvProjectOnObjectRequestSize() described below. Depending on the encryption type the key may be decrypted before it's returned. This function is designed to be used by CSP client programs to access their provisioned keys. It is thread safe if other provisioning operations are taking place at the same time.

The prototype for this function is:

```
UTV_UINT32 UtvProjectOnObjectRequest( UTV_INT8 *pszOwnerName,
UTV_INT8  *pszKeyName, UTV_BYTE  *pubIDBuffer, UTV_UINT32
uiIDBufferLength, UTV_BYTE  *pubKeyBuffer, UTV_UINT32
```

uiKeyBufferLength, UTV_UINT32 *puiKeySize,UTV_UINT32 *puiEncryptionType)

This function takes the following arguments.

UTV_INT8 *pszOwnerName -

A zero-terminated string describing the owner of the key. This identifier is assigned when the key is installed on the NOC and pre-shared with the CSP client program.

UTV_INT8 *pszKeyName -

A zero-terminated string describing the name of the key. This identifier is assigned when the key is installed on the NOC and pre-shared with the CSP client program. It is unique to the key class and does not typically change when a key is renewed due to revocation or expiration.

UTV_BYTE ***pubIDBuffer** -
Pointer to a buffer to return the key's zero-terminated ID in.

UTV_UINT32 **uiIDBufferLength** -
The length in bytes of the buffer pointed to by **pubIDBuffer**.

UTV_BYTE ***pubKeyBuffer** -
Pointer to a buffer to return the key in.

UTV_UINT32 **uiKeyBufferLength** -
The length in bytes of the buffer pointed to by **pubKeyBuffer**.

UTV_UINT32 ***puiKeySize** -
The size of the key returned in the buffer pointed at by **pubKeyBuffer.**

UTV_UINT32 *puiEncryptionType -
The encryption type of the returned key. The encryption types are described below. These macros are defined in utv-core.h

```
UTV_PROVISIONER_ENCRYPTION_NONE     -no encryption
UTV_PROVISIONER_ENCRYPTION_3DES     -3DES, decrypt before return
UTV_PROVISIONER_ENCRYPTION_AES      -AES256, decrypt before return
UTV_PROVISIONER_ENCRYPTION_RSA      -RSA2048, decrypt before return
UTV_PROVISIONER_ENCRYPTION_CUSTOM   -SoC custom, decrypt before return
UTV_PROVISIONER_ENCRYPTION_3DES_UR  -3DES, returned encrypted
UTV_PROVISIONER_ENCRYPTION_AES_UR   -AES256, returned encrypted
UTV_PROVISIONER_ENCRYPTION_RSA_UR   -RSA2048, returned encrypted
UTV_PROVISIONER_ENCRYPTION_CUST_UR  -SoC custom, returned encrypted
```

This function returns the following values.

UTV_OK –
The key has been found, stored in the provided buffer, and its size has been placed in puiKeySize.

UTV_OBJECT_NOT_FOUND –
A key with the specified owner and name was not found.

UTV_BUFFER_TOO_SMALL –
The key was found, but the buffer provided is too small. The key is *not* returned, but its size is placed in the puiKeySize argument.

UTV_BAD_CHECKSUM –
The key was found, but it failed its local cache CRC verification check. The key is *not* returned. The local cache has been corrupted.

6.2.2 UtvProjectOnObjectRequestSize()

This function returns the size of the named provisioned object. This allows calls to UtvProjectOnObjectRequest() to use the correct sized buffer when actually requesting objects. It is thread safe if other provisioning operations are taking place at the same time.

The prototype for this function is:

```
UTV_UINT32 UtvProjectOnObjectRequestSize( UTV_INT8 *pszOwnerName,
UTV_INT8  *pszKeyName, UTV_UINT32 *puiKeySize )
```

This function takes the following arguments.

`UTV_INT8 *pszOwnerName` –
A zero-terminated string describing the owner of the key. This identifier is assigned when the key is installed on the NOC and pre-shared with the CSP client program.

`UTV_INT8 *pszKeyName` –
A zero-terminated string describing the name of the key. This identifier is assigned when the key is installed on the NOC and pre-shared with the CSP client program. It is unique to the key class and does not typically change when a key is renewed due to revocation or expiration.

`UTV_UINT32 *`**puiKeySize** –
The size of the key specified by the object owner and key names.

This function returns the following values.

UTV_OK –
The key has been found and its size has been returned in *puiKeySize.

UTV_OBJECT_NOT_FOUND –
A key with the specified owner and name was not found.

6.2.3 UtvProjectOnStatusRequest()

This function returns many different status information fields in the status structure passed to it. The parts of the status that are most interesting to the provisioning process are covered here.

The prototype for this function is:

```
UTV_UINT32 UtvProjectOnStatusRequest( UTV_PROJECT_STATUS_STURCT
*pStatus )
```

This function returns the following information in the structure pointed at by the pStatus pointer:

`pStatus->usStatusMask | UTV_PROJECT_STATUS_MASK_REGISTERED`

This flag indicates that the device has registered with the NOC.

`pStatus->usNumObjectProvisioned`

The value of this field indicates the number of objects that have been provisioned.

`pStatus->uiLastProvisioningError`

The value of this field indicates error value returned by the last call to `UtvProjectOnNetworkUp() (described below).` These values are defined utv-core.h and can include:

UTV_OK –

The local cache has been updated successfully.

UTV_INTERNET_DNS_FAILS –

This is typically the value returned when the Internet connection fails.

(many other error values are possible, but the ones listed above are the most typical)

This function returns the following values.

UTV_OK –
Device status has been returned.

UTV_NULL_PTR –
The pointer provided to return status info in was NULL.

6.3 Device Agent System Interface Functions

This section describes the functions that the platform's system must call in order to inform the Agent that certain events that SecureTV is dependent on have taken place. The prototypes for these functions can be found in utv-core.h

6.3.1 UtvProjectOnNetworkUp()

This function is called by the system's supervising program when the network first comes up. It causes the Agent to make contact with the NOC if the last time the NOC was checked was 24 hours or more ago. This function does not return any value because it's assumed to be called by an event handler. The actual NOC contact and local cache management occurs asynchronously in its own thread. Therefore this function call will return almost instantly.

The prototype for this function is:

```
void UtvProjectOnNetworkUp( void )
```

6.3.2 UtvProjectOnForceNOCContact()

This function is called by the system's supervising program when it would like the next operation to force NOC contact instead of waiting for the query interval (usually 24 hours) specified by the NOC. This override is only in effect once. Additional calls to this function must be made to force the device to contact the NOC for subsequent operations.

The prototype for this function is:

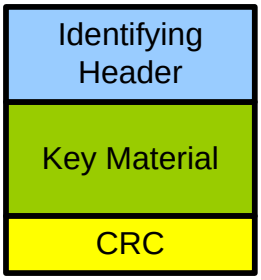
```
void UtvProjectOnForceNOCContact( void )
```

7 Understanding the ULPK

Prior to the availability of SecureTV, a CEM needed to insert device-unique keys in the factory. If there were 4 CSPs in a device there had to be 4 sets of keys inserted at the factory. SecureTV moves factory key insertion to field provisioning of the same keys in the consumer's home. Therefore the factory only relates with a single small, easy to manage UpdateLogic Provisioning Key (ULPK). Each ULPK is device unique and pre-shared with the ULI NOC. The ULPK is used by the Agent during registration so that the NOC can authenticate the device.

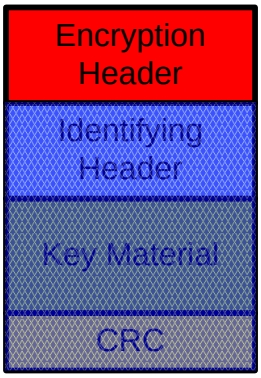
7.1 ULPK Format

Each ULPK is made up of two parts. An ULI-specific core and a SoC-specific encryption header that secures the key. The ULI-specific core has three basic parts. A header that identifies the ULPK as belonging to a particular CEM and device class, key material, and a CRC. The ULPK core is 48 bytes in length.



ULPK Core

This core is encrypted and prefixed with an encryption header that contains SoC-specific instruction for how to decrypt the core. Typically the core plus the encryption header is less than 100 bytes.



Encrypted ULPK Core with SoC-Specific Encryption Header

7.2 ULPK Delivery to the Factory

ULI generates ULPKs, encrypts them using SoC-specific instructions which means they can only be decrypted on the that particular SoC, and uses base64 encoding to create a CSV file that contains hundreds of thousands or even millions of keys. ULI initially sends the factory a ULPK CSV file with about six months worth of keys. The actual number of keys is dependent on the CEM's production estimates. The first CSV file is delivered 90 days in advance of production. Whenever a factory consumes enough keys so that there are only 90 days of keys left in the CSV "reservoir" then the factory must inform ULI and another CSV file will be sent to top off the reservoir.

The CSV file takes the following format. There are commas between each field and CR/LF between each record.

Index	Key	CRC
1717	QgEBAJ0ZAAAQgSAAMAAAADDXY70emLBRZBHzceJYWi6oeHp5IuHiKSVlb5KdxatPx7ttniLLzDxea6AwFG9z+qYyY+IQ0rTHp3WajxMzsI6Erwl	0x5FE43D77
1718	QgEBAJ0ZAAAQgSAAMAAAAEzthLlqYRaeXZdaCVuAXWXkGs330PonS6CQNJKo7btoe1Zob2MQk54nVkvFxRupM3Cp3b52XsPM1GQYJrlmfIO	0x431428BE
...	...	...

ULI is will provide an example ULPK CSV file upon request.

Each ULPK is extracted by the factory, base64 decoded, and inserted as a file into a location in flash that it known to the Agent though the Agent project adaptation layer. Please see the *UpdateTV Agent Quick Start Guide* for more information about how to specify the location of the

ULPK file to the Agent. The factory insertion software should perform a CRC check on each index and key as they are inserted to check CSV file integrity. The factory insertion should read the ULPK back from the device and compare it to the source key to check flash write integrity.

ULPKs do not need to be associated with a particular device serial number. ULPKs may be skipped and discarded. ULPKs should *never* be re-used.

8 Device Provisioning

8.1 Initial Provisioning

When the supervising program on a device first determines that an Internet connection is available it will call the Agent's `UtvProjectOnNetworkUp()` entry point which will automatically register the device, provision it with keys, and check if there are any updates available. This same process takes place automatically if the device is *re-registered* meaning its NOC identity has been reset. After this initial provisioning is complete the keys stored in the local cache are immediately accessible to CSP client programs via the `UtvProjectOnObjectRequest()` interface. The provisioning process is invisible to the supervising program. It is essentially a conversation between the CSP's client program and the Agent. Please see the *SecureTV Interface Specification* section below for more information on SecureTV-related function calls and their arguments.

8.2 Refreshing the Local Cache

The local cache needs to be updated periodically to reflect any key changes that have been made on the NOC. Every time the TV is turned on and it connects to the internet the supervising program calls the Agent's `UtvProjectOnNetworkUp()` entry point. This will force the local cache to be updated from the NOC to reflect any changes in the provisioned keys for that device. This may include the addition of new keys, the deletion of revoked keys, or the renewal of existing keys. The Agent controls when communication with the NOC actually happens to *pace* communication so that only one request for updates occurs every 24 hours. It may be necessary to force the Agent to contact the NOC for trouble-shooting purposes. In this case the *bForceNOCContact* argument to `UtvProjectOnNetworkUp()` should be set to true. Typically a special key sequence or hidden menu is used to initiate a forced local cache update. Normally *bForceNOCContact* should be set to false to avoid network congestion on the NOC.

8.3 Re-Registration

A device may be *re-registered* from the NOC which will force it to delete its network identity and get re-provisioned. This option can be used when a device is reconditioned at the factory to bring it to a known state or by technical support when helping a user with a problem that requires the system to be brought back to its factory state.

8.4 Blacklisting

A device may be *blacklisted* from the NOC which denies it Internet services from ULI including updating and provisioning. Please note that blacklisting does not effect the device's connection to the Internet. When a device has been blacklisted on the NOC the next time the supervising program calls `UtvProjectOnNetworkUp()` the local cache will be deleted. Therefore any CSP client's requesting access to provisioned keys will be denied. There is no way for a device to be taken off the blacklist without a change to its status on the NOC. If a device is taken off the

blacklist it will provision itself again automatically. Blacklisting is used when a given serial number is suspected of theft of service or any other reason that it requires it to be denied service from the NOC. Please note that if the CSP's client caches the provisioned keys the Agent will not be able to prevent them from using it.

9 Managing Keys on the NOC

This section describes how keys are managed on the NOC. This includes key installation, expiration, and revocation. All of these operations are controlled by ULI staff. They are not accessible to CEMs or CSPs.

9.1 Key Installation

9.1.1 Installing Pre-Existing Keys

In cases where the keys are generated outside of ULI because the DRM provider does not share their generation process or the keys cannot be generated for any other reason they will be imported into the NOC in a process that includes encryption and linking the credentials to an expiration date, name, owner, and serial number.

9.1.2 Installing ULI-Generated Device-Unique Keys

In cases where ULI can generate the keys the importation process is exactly as described above except the keys are generated by ULI. ULI has commercial experience generating many different types of keys from X.509 certs to high-entropy cryptographically random key material to special serial numbers.

9.1.3 Installing A Single Pre-Existing Key To A Device Class

A single key can be assigned to an entire class of devices. In this case all devices in the given class will receive the same key. The NOC supports a mode where the key it installed as a file and assigned to a device class along with its expiration date and owner and key name. When provisioning requests are made by a device this key is delivered to all devices in the class. Device common keys do not have serial numbers.

9.2 Key Expiration

The NOC ages the keys that have been installed and checks to see if they're approaching their expiration date. When a class of keys reaches 90 days from its expiration date an e-mail is sent to the ULI network operations staff warning them that expiration date is approaching. If a replacement credential is installed it will immediately be staged for re-provisioning across all the devices in the class and the devices will be notified that they should initiate re-provisioning. If a decision to allow the credential to expire is made then nothing will be done and the credential will eventually expire and be deleted from the device's local cache. Many CSPs choose not to provide expiration dates for their keys to simplify their management. Both the CEM and the CSP are notified by ULI when an expiration date is approaching and coordinated approach is taken to manage the situation.

9.3 Key Revocation

In the case of a compromise to an already provisioned key the NOC can be instructed to revoke the credential. The process that follows is similar to the one described for expiration. A replacement key can be provided or a decision can be made to tell the Agent to delete the key from storage. Key revocation is done in close coordination with the CEM and the CSP. ULI never initiates key revocation on its own.

9.4 Key Addition Threshold

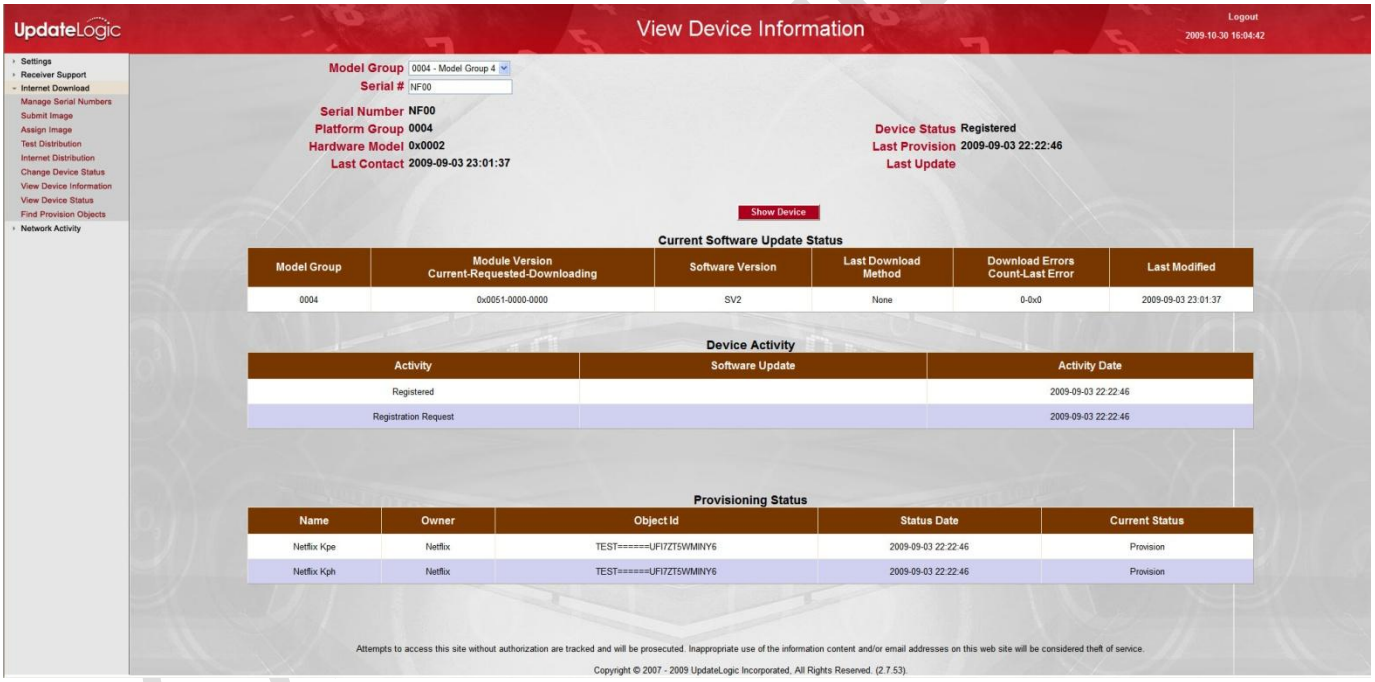
If the pool of device-unique keys reaches a pre-set threshold the NOC sends an e-mail to ULI network operators to alert them that more keys need to be added to the pool.

10 Monitoring Key Status on the NOC

This section describes how key status is checked on the NOC. All of these operations are accessible by CEM staff.

10.1 View Device Information

The View Device Information page on the NOC is located under the "Internet Download" menu option on the main page. Once a device serial number is entered the "Show Objects" button is pressed and the device's registration status, last contact, last provision, and software update status and history, and provisioning history is displayed.



10.2 Find Provision Objects

The Find Provision Objects page on the NOC is located under the "Internet Download" menu option on the main page. It accepts either a device serial number or Object Id (key serial number). When one of these identifiers is entered and the "Show Objects" button is pressed all of the provisioned objects associated with the device that the identifier locates are displayed.