CHAPTER 10

Intel Identity Protection Technology: the Robust, Convenient, and Cost-Effective Way to Deter Identity Theft

People need to be more aware and educated about identity theft. You need to be a little bit wiser, a little bit smarter and there's nothing wrong with being skeptical. We live in a time when if you make it easy for someone to steal from you, someone will.

—Frank Abagnale

Most people have received scam e-mails that prompt them to visit fake web sites that resemble actual bank web sites. If the user is fooled into believing in the "phishing" web site and enters his username and password, then the credentials will be saved by attackers that will later log in to the victim's bank account and drain the account. Besides phishing, an advanced attacker may also infect the victim's computer with key logger malware to capture and record the keystrokes when the victim is typing his username and password for login.

With the rapidly increasing threats of identity theft in today's mobile era, multifactor authentication is deployed more widely than ever. Naïve single-factor username and password combination is likely not secure enough for authenticating access to high-value assets, even though the password is long and complicated.

Multifactor schemes would mitigate phishing and key logging attacks by requiring additional credentials during the authentication process. The username and password compromise the first factor—"something you know." Two other types of credentials are the following:

- *"Something you are"* refers to something that is part of you, commonly your biological characteristics, such as fingerprints. For example, the iPhone 5s is equipped with a fingerprint identity sensor. A user can unlock the phone by scanning his fingerprint.
- *"Something you have"* refers to a physical object that belongs to you. It can be as simple as a "key-card matrix" on which a fairly large number of index-key pairs are printed. During authentication, the web site challenges the user with a randomly selected index, and the user looks up the matrix and enters the corresponding key to sign in. This solution is not ideal, because the same set of keys is repeatedly reused, and may be monitored and replayed by thieves. A more robust "something you have" is a hardware digital token or key fob that displays a one-time password.

The security and management engine is a critical functional component of Intel Identity Protection Technology[1] (IPT). The Intel IPT provides a strong, convenient, and cost-effective solution for multifactor authentication, as well as other features, such as the protected transaction display (PTD). This chapter is dedicated to revealing how the engine takes advantage of its built-in infrastructure to make the IPT possible.

One-Time Password

In contrast to a regular password that is valid for an unlimited number of authentication sessions until it is reset, a one-time password (OTP) is a credential that is used only once. Although the value of an OTP may seem random, it is not randomly generated, but cryptographically derived. A good OTP algorithm shall render it practically infeasible to predict future OTP values based on previous observations. The OTP is usually updated at fixed internals, for example every 30 or 60 seconds, depending on security models of specific applications.

The token (client) possessed by the user and the back-end authentication server are always in sync—they refresh the OTP by performing the same calculation at the same time with the same "derivation materials." In other words, after initialization, the token and the server will both assume the same OTP at any given moment in the future. To prove his ownership of the token to the server, the user types in the OTP value displayed on the token at the time of authentication to satisfy the requirement of second-factor authentication, supplementing other factors (for example, the username and password).

The utilization of OTP significantly increases the difficulty of phishing attacks. The attacker's fake web site has to also collect the OTP entered by the user. Because the OTP is valid for only a small duration, the attacker cannot save the OTP and make use of it later. Instead, the phishing server has to be set up as a real-time man-in-the-middle, where it simultaneously establishes two connections, one with the victim's client platform and the other with the real authentication server. This makes the attack more complex and expensive. Figure 10-1 shows a real-time man-in-the-middle attack scenario.

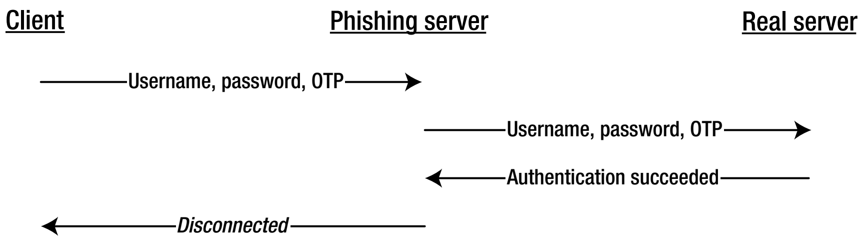


Figure 10-1. *Man-in-the-middle attack*

An OTP system has two aspects to consider:

- *Secrecy*: With reasonable resources, adversaries shall not be able to calculate or guess OTP values. Theoretically, any collision-free one-way cryptography function with a secret *seed* as input is qualified. The HMAC[2] (*hash-based message authentication code*) algorithm is the most popular choice.
- *Synchronization*: The same OTP value must be known by the server and the client at any given moment, without requiring synchronizations after initialization.

The security strength of an OTP system solely depends on the seed, so the seed must be reliably protected by both the server and the client from leakage. Traditionally, more attention has been paid to designing physically strong and tamper-resistant tokens. However, the server's security hardenings are even more critical because if it is hacked, then likely seeds for all tokens are at risk.

RSA Security, the Security Division of EMC, designs and manufactures a well-known OTP token, SecurID. In March 2011, the company issued an open letter[3] stating that its corporate security systems had identified an "extremely sophisticated cyber-attack" being mounted against it. The letter did not disclose technical details, probably due to the concern of benefiting potential attackers, but it revealed that the attack had resulted in *certain information* specifically related to SecurID being extracted from RSA's systems. In the aftermath, RSA offered token replacements or free security monitoring services to its more than 30,000 SecurID customers. The breach cost EMC $66.3 million, according to the company's earnings.

The most famous OTP standards are the *HMAC-based one-time password* (HOTP[4]) and the *time-based one-time password* (TOTP[5]).

HOTP

The derivation algorithm chosen by the HOTP method is HMAC-SHA-1. The HMAC *key*, also referred to as *seed*, is a shared secret agreed by the server and the client at the time of initialization. The key may be randomly generated or calculated from a master secret of the server. The key is static for the life cycle of the client and it must be kept secret by both parties against tampering.

An HOTP is calculated as follows:

```
HOTP(key, counter) := Truncate(HMAC-SHA-1(key, counter))
```

In the `HMAC-SHA-1()` function, *counter* is the data to be hashed. The `Truncate()` function reduces the 160-bit keyed-hash result to a smaller size so the user can conveniently enter the HOTP on a keyboard.

The two input parameters, *key* and *counter*, are the derivation materials. They are used for providing secrecy and synchronization, respectively. After a successful authentication, both the server and the client increment the *counter* by one, hence the *counter* should always be in sync. The server automatically increments the *counter* once it verifies the HOTP. On the client side, for a connected token (such as via a USB port), the connected computer can programmatically increment the counter.

However, many token products are not equipped with connection capability. The advantage of a connection-less token is obviously its simple hardware and low BOM (*bill of materials*) cost—it needs only small tamper-resistant storage for the *key* and *counter* and an HMAC-SHA-1 logic; it does not require circuits for USB or clocking. The tradeoff is that the user has to, after a successful authentication, manually notify the token and have it increment the *counter* and generate the next HOTP. The notification is usually realized by the user pushing a button on the device. This manual step introduces uncertainty and potential problems for synchronization. For example, the user may accidentally push the button twice, resulting in the token's counter value being more advanced than the server's. To take care of such issues, the HOTP protocol defines a "look-ahead window," where the server calculates the next *s* HOTPs. The authentication is accepted as long as any of the *s* HOTP matches the HOTP received from the client. The window size *s* cannot be too large, otherwise security may be compromised.

But this mechanism does not completely resolve all potential synchronization issues. Imagine the user's three-year-old child plays with the token and pushes the button countless times. The server's look-head window will not cover this case, and the token must be returned to factory for reinitialization.

TOTP

The TOTP scheme is a variant of HOTP that replaces the *counter* in the HOTP with a time value, *time*:

```
TOTP(key, time) := Truncate(HMAC(key, time))
```

The HMAC function may be HMAC-SHA-1, HMAC-SHA-256, or HMAC-SHA-512. The *time* is equal to the Unix time or Epoch time (number of seconds that have elapsed since midnight UTC (coordinated universal time) of January 1, 1970) divided by a predefined interval, with the default floor function. The `floor(x)` function represents the greatest integer that is not greater than fraction *x*. The recommended interval is 30 seconds:

```
time := floor(Unix_time/interval)
```

Compared to the HOTP, the TOTP scheme uses *time* as the counter for synchronization, which eliminates the problems of incrementing the counter for connection-less tokens. The TOTP scheme requires a token to have clocking capability by embedding an oscillator in the device. A token's clock drift needs to be considered and accommodated accordingly by the server. The protocol also recommends the server to implement "look-ahead" and "look-behind" windows to for resynchronization when a tolerable amount of clock drifts have occurred on the token.

The TOTP scheme is the cornerstone of the reference architecture of OATH[6] (*Initiative for Open Authentication*), an industry-wide collaboration to promote the adoption of strong authentication.

Transaction Signing

The OCRA[7] (*OATH* Challenge-Response Algorithm) is an authentication and signing mechanism created by the OATH. The OCRA algorithm is based upon HOTP with extension to including various types of information in the calculation of the OCRA.

In a nutshell, the calculation of OCRA uses the following formula:

```
OCRA := CryptoFunction(Key, DataInput)
```

The same formula is applicable to both the server and the client. The `CryptoFunction` defines the HMAC algorithm (HMAC-SHA-1, HMAC-SHA-256, or HMAC-SHA-512) and the result size after truncation. *Key* is a preshared secret, like in the HOTP scheme. The value *DataInput* is a concatenation (denoted by symbol "||") of the byte arrays of a number of variables:

```
DataInput := OCRASuite || Counter || Q || P || S || T
```

Here's what these variables mean:

- *OCRASuite*: Represents the suite of operations to calculate the OCRA. The OCRASuite string describes the selection of `CryptoFunction` and the list parameters that are included in the DataInput following OCRASuite.
- *Counter*: A 64-bit unsigned integer that is initialized to 0. It is incremented by both the server and the client after every successful authentication session.
- *Q*: The 128-byte challenge sent from the other party.
- *P*: Digest of a password preagreed by the server and the client. The hash algorithm can be SHA-1, SHA-256, or SHA-512.
- *S*: Contains application-specific information of the current session, up to 512 bytes.
- *T*: Current timestamp.

The *Counter* is optional. Some other parameters (*P*, *S*, and *T*) may also be absent as defined by the OCRASuite. A typical OCRA authentication session in which the server verifies the client's identity is a three-way handshake, as shown in Figure 10-2.

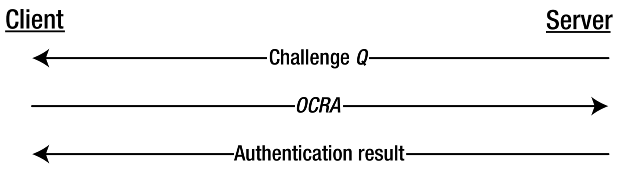


Figure 10-2. *Three-way handshake for one-way OCRA authentication*

Four modes are defined for OCRA:

- *One-way authentication*: The server verifiers the client's identity by sending a challenge *Q* to the client and verifies the OCRA value received from the client. This is the usage depicted in Figure 10-2.
- *Mutual authentication*: The server and the client verify each other's identity by exercising the one-way authentication in both directions. The client verifies the server's identity first. See Figure 10-3 for the four-way handshake flow.

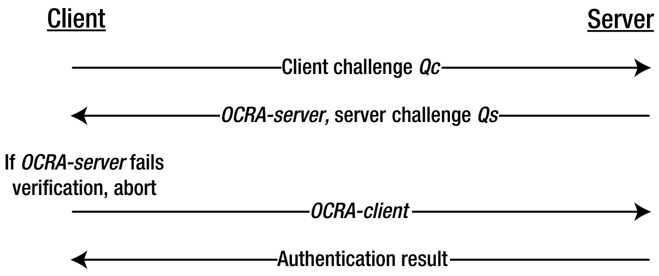

Client
Server
Client challenge Qc
OCRA-server, server challenge Qs
If OCRA-server fails verification, abort
OCRA-client
Authentication result

Figure 10-3. *Four-way handshake for mutual OCRA authentication*

- *One-way signature*: Similar to one-way authentication, but session information *S* is not used in calculating the OCRA.
- *Mutual signature*: Similar to mutual authentication, but session information *S* is not used in calculating the OCRA.

Besides OCRA, it is also possible to employ asymmetric-key cryptography and public key infrastructure (PKI) to achieve the same authentication and transmission integrity. The advantage of OCRA is its simper cryptography logic (HMAC) and faster computation. Public key algorithms require more gates to implement and more clock cycles to compute, which poses challenges for BOM cost and performance for small form-factor client devices. Now that the server and the client already have a shared secret, the OCRA makes use of it to avoid the higher-cost and inefficiency of PKI.

Using OCRA, the man-in-the-middle attack presented in Figure 10-1 is no longer a threat, because the transaction, including the session-specific information (for example, the recipient of a money transfer), is signed with a key that is known by only the server and the client. Consequently, the attacker is not able to alter a legitimate user's transactions or initiate his own transactions, because he cannot forge signatures without knowing the correct *key*.

OTP Tokens

Numerous two-factor authentication solutions that deploy OTP as the second factor are on the market today. An OTP token can be implemented in software or hardware.

Functioning as an OTP client, a software or virtual OTP token is a program installed and executed on a desktop computer or a mobile device. The software OTP has a number of pros:

- *Low cost*: No hardware purchase required.
- *Convenience*: No hassle of carrying physical tokens. No worries about replacing the token when its battery runs out.
- *Transparency*: In most cases, the user does not need to type in the OTP. The software automatically calculates and transmits it to the server.
- *Reliable synchronization with server*: Clock drifting on a computer is much less of a concern than clock drifting on a small token device, because the computer's clock always synchronizes with the time server over the network.
- *Easy reinitialization*: When reinitialization is necessary for any reason, there is no need to return the token to the vendor. Reinitialization with the server can be done remotely.

While enjoying these advantages, the software solution has a key drawback—it is more vulnerable. Generally speaking, because software OTP may be compromised by malware installed by viruses or remote attackers, the robustness of software OTP cannot match that of well-implemented hardware OTP systems. Physical access and special equipment are required to tamper a hardware OPT device, making the attack more difficult and costly to mount. For enhanced security, hardware OTP clients are deployed by many large enterprises and government agencies. The pros of the software tokens are exactly the cons of hardware tokens.

Is it possible to feature the pros of both software and hardware OTPs? The security and management engine provides such a solution for the Intel IPT.

Embedded OTP and OCRA

The second factor in multifactor authentication—"something you have"—does not have to be a separate object. It can be the computer that the user is operating on. In Intel's IPT solution, the security and management engine that is built in an Intel platform is the second factor. Security-wise, the engine is physically a hardware device that the user carries with his computer; therefore, its protection strength is comparable to hardware OTP tokens. On the other hand, thanks to its embedded nature, it has all the desirable properties of software OTP as well.

The OTP scheme supported by the engine implements the TOTP algorithm and the OCRA protocol. The solution is compliant with the OATH standard. This standard-based model that Intel IPT uses simplifies interoperability with other third-party components.

Token Installation

The installation (also referred to as *provisioning*) of a token on the embedded engine is equivalent to a hardware token's manufacturing process. As required by the TOTP, two pieces of information are delivered to the client from the server during the installation process:

- *Key*: As defined in the TOTP calculation formula.
- *Time baseline*: The Unix time at the moment of installation.

Obviously, the transmission of *key* must be encrypted because it is the root of security for all upcoming authentication sessions. The transmission of *time baseline* and encrypted *key* should be integrity protected to prevent unauthorized alternation by denial-of-service attacks.

As discussed in Chapter 3, the engine's kernel lacks the knowledge of wall-clock time, but it is capable of securely tracking time that has elapsed for individual applications. Therefore, the server has to send *time baseline* to the engine during provisioning. The OTP application calls the *set time* kernel function immediately upon receiving the baseline from the server. When an authentication is requested, the OTP calls the kernel's *get current time* function and uses the returned value to calculate the TOTP.

Recall the EPID (enhanced privacy identification) algorithm and the SIGMA (SIGn and Message Authentication) protocol introduced in Chapter 5. They are the backbone of the OTP provisioning process. In the SIGMA session, the authentication server is the verifier and the OTP client is the prover. The service provider must be issued a verifier certificate beforehand. The server's certificate chain is verified by the embedded engine during the SIGMA session. The conceptual provisioning flow is illustrated in Figure 10-4.

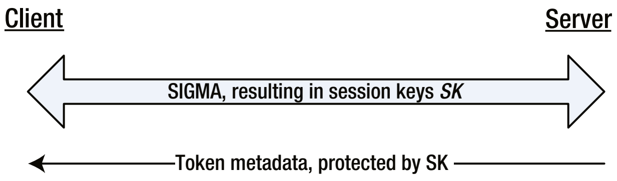

Client
Server
SIGMA, resulting in session keys SK
Token metadata, protected by SK

Figure 10-4. *Conceptual OTP provisioning flow*

Refer to Figure 5-7 in Chapter 5 for details on the SIGMA messages. At the end of a successful SIGMA session, the server has assured that the client is an IPT-capable Intel platform and the client has confirmed that the server is a valid authentication server that supports Intel IPT. Both parties also have derived the shared session keys (*SK*), including an encryption key and an integrity key. The token metadata, including OTP *key* or *seed*, choice of the HMAC algorithm, current time, and so forth, is delivered from the server to the client securely with the protection of *SK*. The client saves the metadata in secure nonvolatile storage. It can either store the data on the flash chip by invoking the kernel's secure storage capability, or it rewraps the data using DAL (*dynamic application loader*; see Chapter 9 for details) application-specific keys and sends to the host for storage. In the latter case, the data is transmitted back to the engine when an OTP is requested.

Although the provisioning can ideally be a once-in-a-lifetime event, it is necessary to reprovision the token under certain circumstances. For example, if the platform's RTC (real-time clock) well is reset due to reasons such as a drained coin-cell battery, then the secure timer installed by the OTP application will be lost. The OTP firmware has to request the server to install the token again. On the other side, the server may also request reprovisioning; for example, in case the *seed* is compromised. The easy provisioning process is a major advantage of the Intel OTP solution, compared to physical token systems.

TOTP and OCRA Generation

The TOTP and OCRA generation flow is straightforward. After receiving a generation request from the host via HECI (host-embedded communication interface), the firmware reads the current time from the kernel and performs the calculation using *time* and *key*. For OCRA, data input parameters such as the server's challenge and session information are provided by the host to the firmware together with the generation request.

The resulting TOTP or OCRA is sent to the host in the clear for authentication with the server. Note that if the token metadata is stored on the host, then it must be loaded to the embedded engine first. The firmware has no direct connection with the server, and all communication is proxied by the host application.

Highlights and Lowlights

Let's summarize the attractions of Intel's embedded OTP solution:

- *Strong protection*: As a module of the security and management engine, the OTP inherently benefits from the comprehensive hardening measures (refer to Chapter 4) implemented on the engine. The protection is rooted in tamper-resistant hardware, meeting or exceeding the security of consumer-grade hardware tokens.
- *Low cost*: To benefit from the technology, the user does not need to purchase new hardware or software. Almost all service providers support the feature at no cost to customers. Compared to using hardware tokens, the cost of initial setup and continuous management for Intel's OTP is considerably less, which is an incentive for deployment.

- *Multiple tokens in one device*: No more physical tokens. No worrying about damaging or losing your tokens. Token theft is also a much lesser risk now because stealing a computer is more difficult than stealing a small token device. Furthermore, more than one token can be built in a single device—your computer. It is unimaginable to carry many tokens with you all the time.
- *Transparency*: After the initial setup, the second-factor authentication happens in the background without human interaction. The user signs in to web sites or networks by entering merely a username and password combination, just as he did before.
- *Revocable server*: The EPID infrastructure, backed by the engine and Intel's back-end server, ensures that a compromised authentication server can be revoked. Besides compromise, the authentication server may be revoked due to other predefined reasons.

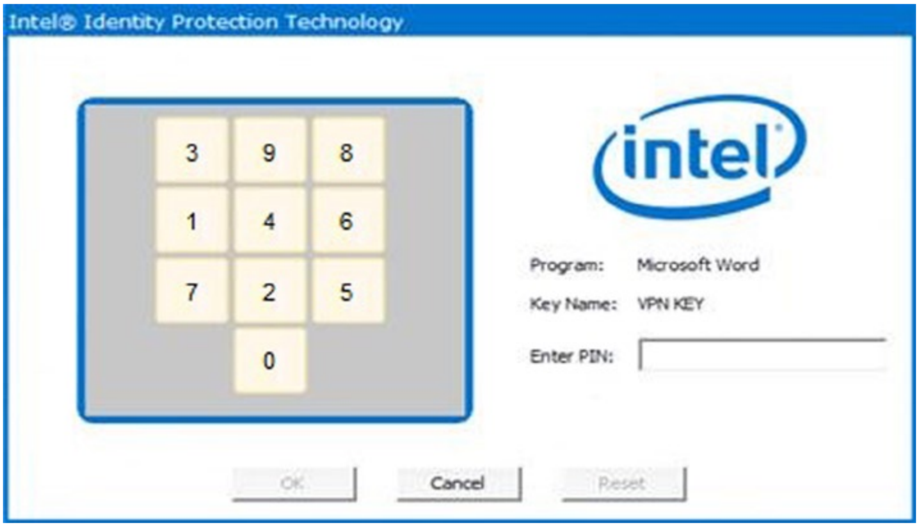
In the meantime, because the tokens, once provisioned, are tied to the hardware of a specific platform, it is impossible to invoke the same token on different devices. As a result, if the user needs to log in to his bank account from more than one device (for example, from both his laptop and tablet), then all devices must register with the bank's web site and install a token, respectively. Fortunately, the inconvenience is trivial, because the provisioning is supposedly a one-time procedure for a device.

However, when the user occasionally has to log in from a public or someone else's computer, an alternate second factor must be utilized in lieu of the Intel OTP token. Service providers must offer feasible backup approaches for the second factor; for example, sending a verification code to the user's cellphone or e-mail address and having the user enter it for authentication.

Protected Transaction Display

The PTD is another critical ingredient of Intel IPT. It is introduced for a different usage model from the OTP and can be incorporated with the OTP. The PTD is designed to enable reliable collection of the user's confirmation or PIN (personal identification number) entry, and detection of malware's falsification of the user's input. The PTD can also ensure that a PIN entry is securely transmitted from the Intel platform to the authentication server, shielded from illegal eavesdropping.

The uniqueness and core innovation of the PTD that distinguishes it from other solutions is that it leverages the PAVP (protected audio and video path; see Chapter 8 of this book for details) technology of Intel platforms to display the authentication sprite, such as the PIN pad, on the user's monitor. Because the PAVP isolates the sprite and protects it from being accessed by the host, malware running on the host operating system is not able to see the sensitive overlay area. As a result, software cannot fake a user's mouse clicks or scrape the screen. Figure 10-5 shows what a user sees during a PTD session.



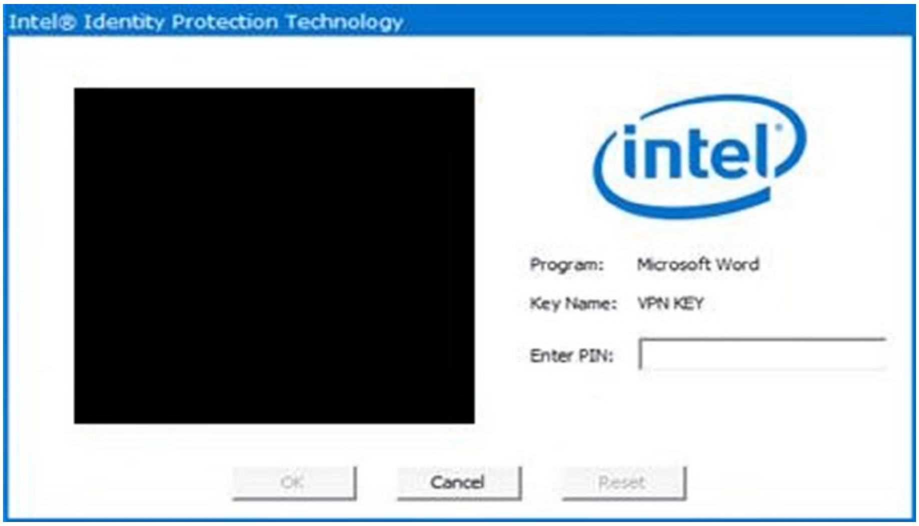
Program: Microsoft Word

Key Name: VPN KEY

Enter PIN:

Figure 10-5. *PAVP-protected PIN pad on the end user's screen*

The user uses his mouse to click the secure PIN pad to enter the PIN. An entry is represented by an asterisk. To further enhance security, the location of the dialog box overlay is randomized for an authentication session; the positions of the ten digits of the PIN pad are also randomized every time. The PIN pad area is not visible to the host. Figure 10-6 shows what an attacker's screen scraper would capture.



Program: Microsoft Word

Key Name: VPN KEY

Enter PIN:

Figure 10-6. *PIN pad captured by a screen scraper*

Drawing a Sprite

Several flows and designs can be utilized when drawing a sprite. Figure 10-7 presents a sample in which a remote authentication server draws the sprite with the assistance of PAVP. In the diagram, the IPT proxy is the IPT's software component running on the host operating system. GPU stands for *graphics processing unit*. Symbol (*data*)*k* denotes the ciphertext of cleartext *data* encrypted with an AES[8] (*advanced encryption standard*) key *k*.

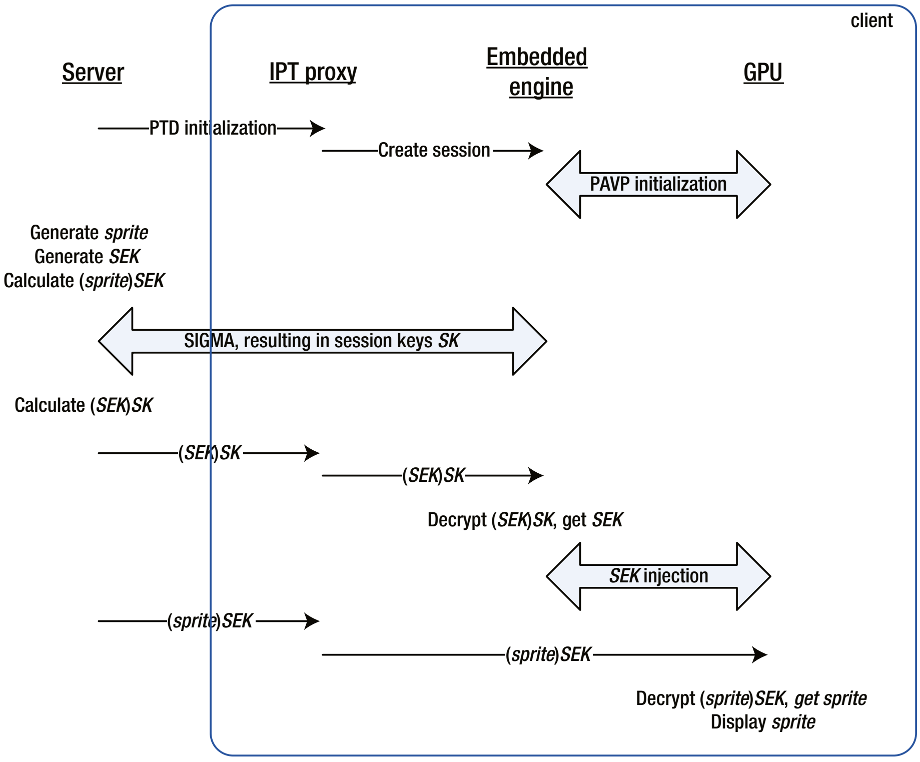


Figure 10-7. *Authentication server drawing a secure sprite*

Similar to OTP, the authentication server must be an endorsed verifier of the SIGMA protocol. To draw a secure sprite on the screen, the server starts with requesting the IPT to initialize a PAVP session. The server then creates the sprite and encrypts it with a randomly generated sprite encryption key or *SEK*. Next, a SIGMA session is established between the server (verifier) and the client (prover). The SIGMA yields the session key *SK* shared between the server and the firmware. The server wraps *SEK* with *SK*. The resulting (*SEK*)*SK* is delivered to the embedded engine, which in turn decrypts and recovers *SEK*. Next, the engine injects *SEK* to the GPU. On the host side, the IPT proxy receives the encrypted sprite from the server and passes it to the GPU for rendering and display.

The sequence assures that the clear sprite is never exposed during the entire transmission path from the server to the GPU. Attackers hidden on the Internet or the host can see only the encrypted version of the sprite; they cannot access the encryption key, thanks to the security provided by the SIGMA protocol. The graphics kernel code, like shaders, loaded by the host driver, cannot access sprite frames either. The security is safeguarded at the hardware level.

Gathering the User's PIN Input

The IPT proxy is responsible for collecting the user's clicks on the secure PIN pad. Notice that, as software, the IPT proxy has no knowledge of the position of the PIN pad or its digit button layout, hence it is not capable of calculating the user's PIN input. The proxy records the coordinates of the clicks and reports to the authentication server. The server is the only entity that is able to interpret the user's input. It does so by comparing the click coordinates with the sprite it created.

Optionally, the coordinates may be encrypted using a SIGMA session key before transmission. Figure 10-8 shows a sequential diagram of this flow.

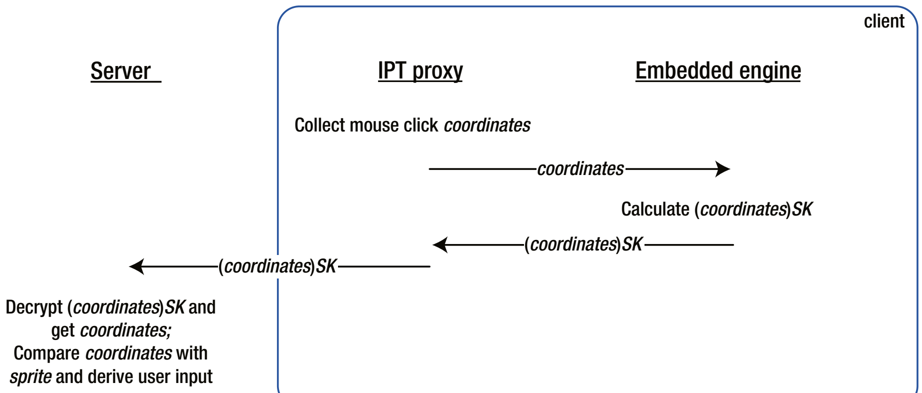

client
Server
IPT proxy
Embedded engine
Collect mouse click *coordinates*
coordinates
Calculate (*coordinates*)*SK*
(*coordinates*)*SK*
(*coordinates*)*SK*
Decrypt (*coordinates*)*SK* and get *coordinates;* Compare *coordinates* with *sprite* and derive user input

Figure 10-8. *Authentication server gathering coordinates of mouse clicks and deriving user input*

Firmware Architecture

Depending on product, the IPT may be implemented as an applet for the engine's DAL feature, or a native firmware module on the engine. If the firmware supports DAL, for example, on most Intel Ultrabook models, then the IPT implementation will be distributed in a Java applet. On certain smartphones and other products where the DAL is not built into the engine's firmware, the IPT will be a native firmware ingredient that is loaded from the system's flash chip. The firmware design and functionalities of the IPT component are identical for both variants.

Figure 10-9 illustrates the high-level firmware architecture. Internal to the security and management engine, the IPT, together with the DAL that loads it, reside in a dedicated task. The IPT/DAL task is not consumed by any other components of the firmware. The IPT/DAL task consumes the following components of the engine to realize its IPT functionality:

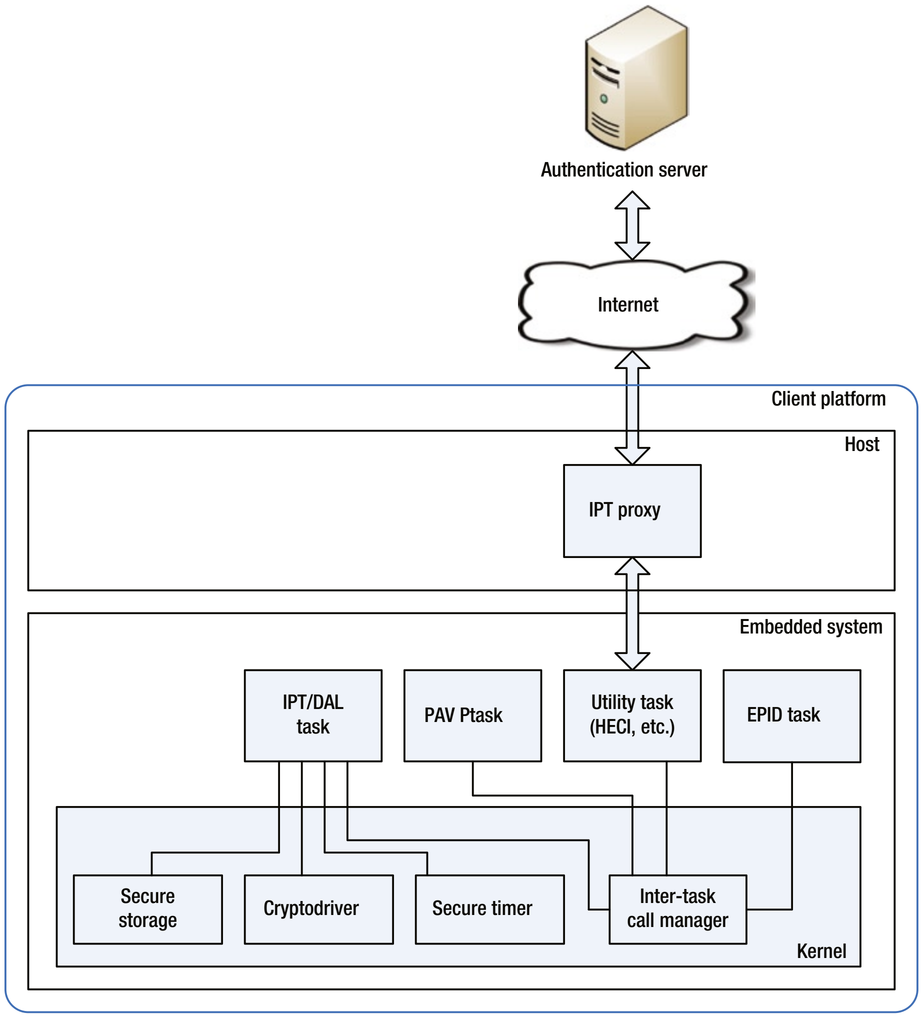


Figure 10-9. *Architecture of the embedded IPT*

- *Storage manager*: The metadata of installed tokens can be stored on the flash device in the engine's data partition. If the metadata is stored on the host, then the application-specific encryption and integrity keys for protecting the metadata are stored on the flash.
- *Cryptography driver*: The TOTP and OCRA calculations use HMAC; the PTD uses AES and HMAC. Additional IPT features may use other cryptography algorithms, such as RSA[9] (*Rivest, Shamir, and Adleman*) digital signing.
- *Protected runtime clock*: The TOTP calculation requires a timestamp.
- *PAVP task*: The PAVP is used by PTD for displaying secure sprites.
- *EPID task*: The provisioning of a token and the session initialization for PTD require EPID and SIGMA's prover functionality.
- *Utility task*: For the HECI communication with the IPT proxy running on the host. Note that HECI is the only interface of the IPT firmware. The IPT does not consume DMA (direct memory access) or network interfaces.

Embedded PKI and NFC

Intel continues to develop innovative technologies to safeguard users' identity. Recently, to further enrich IPT, two new members, PKI and NFC (near-field communication), have joined the IPT family.

The PKI feature supports secure nonvolatile storage for users' asymmetric private keys, such as an RSA key, in the security and management engine. Equipped with this solution, computers can be seamlessly integrated with existing usages, such as VPN (virtual private network) authentication, e-mail and document signing, and so on. Once a private key is imported to the engine or generated by the engine, it will never be exposed in the clear to the external world and all cryptography operations with the private key are performed inside the engine.

The NFC feature enables a user to pay for his online purchases by simply tapping an NFC-capable credit card against the NFC sensor and the secure element chip in the computer, and completing the transaction with positive identity authentication. Thanks to NFC, the customer no longer has to manually type in the long 16 digits of the credit card number. The solution is not only more convenient and user-friendly but also more secure. Key logger malware is not able to steal the card number because it is not entered through a keyboard. The credit card information is processed by the security and management engine and securely transmitted to the server, with robust hardware-level protection.

For more technical details of Intel IPT's PKI and NFC features, refer to the white paper[10] "Deeper Levels of Security with Intel Identity Protection Technology." More information about Intel IPT can be found on the official web site (`http://ipt.intel.com/`).

References

1. Intel Identity Protection Technology, http://ipt.intel.com/, accessed on April 20, 2014.

2. National Institute of Standards and Technology, "The Keyed-Hash Message Authentication Code (HMAC)," http://csrc.nist.gov/publications/fips/fips198-1/FIPS-198-1_final.pdf, accessed on November 17, 2013.

3. RSA, the Security Division of EMC, "Open Letter to RSA Customers," www.sec.gov/Archives/edgar/data/790070/000119312511070159/dex991.htm, accessed on May 10, 2014.

4. Internet Engineering Task Force, "HOTP: An HMAC-Based One-Time Password Algorithm," *Request for Comments 4226*, http://tools.ietf.org/html/rfc4226, accessed on May 10, 2014.

5. Internet Engineering Task Force, "TOTP: Time-Based One-Time Password Algorithm," *Request for Comments 6238*, http://tools.ietf.org/html/rfc6238, accessed on May 10, 2014.

6. Initiative for Open Authentication, www.openauthentication.org/, accessed on May 10, 2014.

7. Internet Engineering Task Force, "OCRA: OATH Challenge-Response Algorithm", *Request for Comments 6287*, http://tools.ietf.org/html/rfc6287, accessed on May 10, 2014.

8. National Institute of Standards and Technology, Advanced Encryption Standard (AES), http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf, accessed on November 17, 2013.

9. RSA Laboratories, "PKCS #1 v2.1: RSA Cryptography Standard," ftp://ftp.rsasecurity.com/pub/pkcs/pkcs-1/pkcs-1v2-1.pdf, accessed on November 17, 2013.

10. Intel Identity Protection Technology white paper, "Deeper Levels of Security with Intel Identity Protection Technology," http://ipt.intel.com/Libraries/Documents/Deeper_Levels_of_Security_with_Intel%c2%ae_Identity_Protection_Technology.pdf, accessed on May 10, 2014.