CHAPTER 5

Privacy at the Next Level: Intel's Enhanced Privacy Identification (EPID) Technology

The fantastic advances in the field of electronic communication constitute a greater danger to the privacy of the individual.

—Earl Warren, 14th Chief Justice of the United States

You've probably clicked the "I agree" button hundreds of times on privacy policy statements of service providers' web sites, from Gmail to Netflix, from Amazon to your favorite game apps. Most people simply want to enjoy the service or access contents as soon as possible, and thus do not bother to read through the privacy policy from beginning to end before giving consent. Many times, people are willing to share their private information with the service providing site/server, and they rely on the vendors' good faith to protect their privacy and not share with third parties.

To build the infrastructure for protecting the privacy of Intel's consumers, Intel invented the enhanced privacy identification (EPID) technology, which is implemented by the security and management engine on big core and systems-on-chip platforms for servers, desktops, laptops, tablets, and smartphones.

Redefining Privacy for the Mobile Age

Service providers normally promise some level of protection in their privacy policy agreements, such as not selling or renting out your personal information (name, gender, date of birth, mailing address, e-mail address, and so forth). At the same time, in exchange for the free or paid services, consumers likely have to allow service providers to archive your activities and push customized marketing correspondence to your mailbox. It may be useful information to you, or, most of the time, may be treated as spam. Notice that privacy policies and options provided to users are often subject to change. New

options that have defaults as not private may be introduced. This puts even the paranoid users at a loss by having to keep up with each new feature update. In addition, it is not surprising that the Google Ads on the websites you browse are promoting products you recently showed interest in. Service providers have good incentives to make the most out of your private information. Monetization of user data is big business and a significant revenue source for many providers, especially social networking websites and apps.

The mobile computing age brings with it increasing risks to users' privacy. There are hundreds of thousands of mobile apps out there, and counting. If you are paying attention to the list of privileges that an app asks for before installation, you will find many require access to data stored on your device, such as personal information, your phone book, call history, text messages, and so forth. A paranoid user may wonder, "Hmm, why does this music app need to access my phone book?" and exit installation. But many do not bother to question. Hence, privacy is at risk.

What is the ultimate and true privacy? The nuanced answer depends on an individual's expectation, which varies based on factors such as type of data, social context, culture, and so on.

A simple answer, however, is *anonymity*. In a perfect world of anonymity, there is no identification. Everyone appears identical. In terms of computer privacy, anonymity can be realized in two ways: passively and actively.

Passive Anonymity

Imagine that an online movie service does not save the list of the contents you have watched, because you do not want others to know what kinds of movies you favor; imagine that a prescription medicine reseller does not record the history of your purchases, because you don't want to expose your health information. This is *passive* anonymity. The realization of such anonymity completely or partially relies on the attitude of the parties you are dealing with. If the movie service wants, it can save the list of titles you have streamed, and even details such as where in the titles you paused or fast-forwarded. Similarly, an online medicine vendor could derive, without much difficulty, what diseases you are suffering by examining the prescriptions you ordered.

One may argue that what types of movies a person likes is not something really secretive. However, no one knows whether such data, seemingly harmless today, could be used against you in the future. In information security, the principle of *least privilege* requires that an entity must be able to access only the information and resources that are necessary for its legitimate purpose. When talking about privacy, least privilege of the service provider is always the best interest to consumers.

An important point in the privacy discussion is the user's expectation, which varies by context. For example, an end user may be okay with sharing the list of movies he has watched with his personal friends on social networks, but not with his work colleagues. Sometimes the user may want to watch some movies privately without anyone else knowing.

Practically, end users cannot rely on the service providers' good faith to protect their privacy. The bottom line is that users' privacy is not in the providers' interest—and may even be against their interest. And even the definition of "privacy" is often up to the providers' discretion. You may choose to believe that some big names are not evil, but you simply cannot trust everyone. The privacy commitment had better be enforced by technology, and not human beings. This is *active* anonymity.

■ **Note** Passive anonymity relies on human beings for enforcement.

Active Anonymity

In contrast to passive anonymity, active anonymity does not depend on human beings for enforcement. The anonymity is natively built in the technology.

■ **Note** Active anonymity relies on technology for enforcement.

There are two fundamental and functional problems to resolve by an active anonymous authentication technology:

- *Authentication*: The user must prove to the service provider that the user is eligible to receive the service. Eligibility is established by showing that the user is a member of a group that is granted access to the specific service. There are various different criteria for becoming a member of the group—for example, possessing a specific hardware device or paying a service fee. The criteria of becoming a group member are defined by the service provider.
- *Anonymity*: The service provider must not be able to trace the user. The key words here are *not be able to* rather than *do not*. In other words, even if the service provider wants to identify a user, it should not be possible to. The user only has to show that he/she belongs to the group of eligible users, without revealing identity. Since all users in the group are allowed to receive the service, this user's request should be fulfilled after being authenticated. The technology must be designed to disable the service provider from possibly distinguishing any individual users from any other users in the same group. Furthermore, the technology must provide an option to the user so that the service provider cannot tell whether or not two or more service requests were originated from the same user.

How to achieve these two goals of anonymous authentication? A straightforward approach is to have all users that belong to the same group share the same credential for authentication. It is a feasible solution, and is in fact being used by many mobile products in the market today. But is it good enough?

The problems of this credential-sharing design are as obvious as its simplicity. Once any device is successfully compromised by attackers, and credentials are leaked, the security of the entire authentication system of the service provider is broken, and all devices with the same credential are impacted. This is a typical break-once-run-everywhere (BORE) scenario. Dealing with the aftermath is very expensive and

cumbersome. To mitigate BORE, besides the two basic goals of authentication and anonymity, there is a third desired characteristic:

- *Revocation:* The technology must provide the means to revoke a select member of a group or the entire group in order to terminate services for only the select user or group without impacting nonrevoked members and groups. The reasons for revocation are determined by the providers. It could be a user's violation of the service agreement or loss or compromise of credential.

Processor Serial Number

Flashing back to 1999, Intel's Pentium III processor introduced a new feature, known as the processor serial number or PSN. The serial number is a 64-bit string programmed to processor hardware during the manufacturing process. The serial number of a processor is unique to the processor. It cannot be changed or erased during the lifetime of the processor. When the read permission is turned on, software can retrieve the PSN value by simply issuing a `CPUID` instruction. Note that the `CPUID` instruction returns other information, such as processor type and the presence of processor features, in addition to the PSN.

The read permission of the PSN can be enabled or disabled through one of two methods, as described next. The goal of restricting read permission is to make sure that the owner of the platform is aware when the PSN is available to be retrieved and by whom:

- *BIOS:* The manufacturer of the platform should provide users with an option in the BIOS configuration for disabling all software programs and websites to read PSN. Some BIOSes use *enabled* as the default value. This is a more advanced approach, because not all users know how to change BIOS configurations.
- *Control utility:* Intel released a Windows software program that lets end users configure the list of software programs and websites that are allowed to read PSN. The utility is a service that launches when Windows is booting. It is a convenient way for users of all skill levels to manage the exposure of the PSN.

Software or websites reading the PSN without the user's consent pose privacy concerns. Opponents of the PSN expressed concerns about the design of the control mechanism. For example, some BIOSes may not offer an option for users to disable the PSN. Even if such an option is offered, if it is set to *on* by default, then some consumers may not know how to enter BIOS and change the configuration. Fortunately, besides using BIOS, a user can also turn off the PSN in Intel's PSN control utility. However, the PSN may be read by software before the control utility is loaded and functional. Rogue software services that load before the control utility may read the PSN during the Windows boot process and save the value for later use, essentially bypassing the enforcement of the control utility. Furthermore, the control utility is software and may be hacked. Relying on software to protect hardware information that has implications on privacy is not a good security practice.

So why was the PSN introduced in the first place? A number of applications may benefit from the PSN. Here are some examples:

- *Secure authentication*: Take website login, for example. Traditionally, a bank's website only asks for the customer's username and password as login credentials. This is called *one-factor authentication*, where only one factor, namely, "something you know," is required. If the password is acquired by an attacker, then he can successfully log in from his computer. To enhance the authentication security, the PSN can serve as another authentication factor, that is, "something you have." To realize the two-factor authentication, the website checks the PSN of the computer in addition to the username and password. With the PSN enforcement, the attacker will not be able to log in from his computer even if he has stolen the victim's password.
- *Software piracy mitigation*: During installation, software may save the PSN of the platform. At runtime, the software checks the PSN against the stored value and functions only if they match.
- *Corporate computer management*: The PSN makes the resource management tasks of corporates' information technology (IT) departments easier. Replacing the processor on a computer is a much less frequent event than replacing other hardware and software components. Therefore, the constant PSN value allows the IT administrators to reliably identify individual platforms in the corporate network. The status of a platform can be monitored and tied to its processor PSN. Changes to the platform can be easily identified and managed.

Note that in the usages of secure authentication and software theft mitigation, the PSN protection may cause inconvenience to legitimate users. For example, if the user upgrades his processor, then he must re-register the new processor with all software vendors and websites that leverage the PSN, which could be cumbersome.

Admittedly, the PSN mechanism has its functional problems. For example, there is no infrastructure to support PSN revocation. If an attacker is able to steal the PSN of the victim's processor and exploit other vulnerabilities in software or websites, then he can possibly circumvent the two-factor authentication and log in to the victim's bank account.

However, the major concern of the PSN is privacy. Opponents argue that the mechanism for controlling the PSN access is not sufficiently robust to guarantee that the PSN is only available to software and websites that are explicitly allowed by the platform owner. In other words, unauthorized software or websites may be able to retrieve the PSN without the owner's knowledge. Even for authorized entities, there is no governance or enforcement on how they use the PSN. For example, all Internet activities of a user may be traced by misbehaving software. Abusing the PSN would compromise privacy.

Intel cares about consumers' privacy and has been dedicated to protecting it. In response to public's concerns, Intel discontinued the PSN feature in its processor products after Pentium III.

But the general demand for hardware-protected authentication still exists. How to achieve it while safeguarding users' privacy?

EPID

One of the major achievements of Intel's research effort in anonymous authentication is the enhanced privacy identification[1] (EPID). The EPID is a novel technology that resolves all aspects of the active anonymity problem: authentication, anonymity, and revocation.

Mathematically, the EPID is built on finite field arithmetic and elliptic curve cryptography (ECC). Interested readers should refer to the publications listed in the "References" section at the end of this chapter for mathematical details of the EPID.

The EPID ecosystem defines three entities:

- *EPID authority*: Responsible for generating EPID groups and private keys; also responsible for revoking members and groups. It has a root ECC key for signing group public keys, EPID predefined parameters, and revocation lists.
- *Platform*: Usually an end-consumer device that receives services.
- *Verifier*: Usually a service provider that provides premium services for the specific device.

The relationships among the three entities are shown in Figure 5-1.

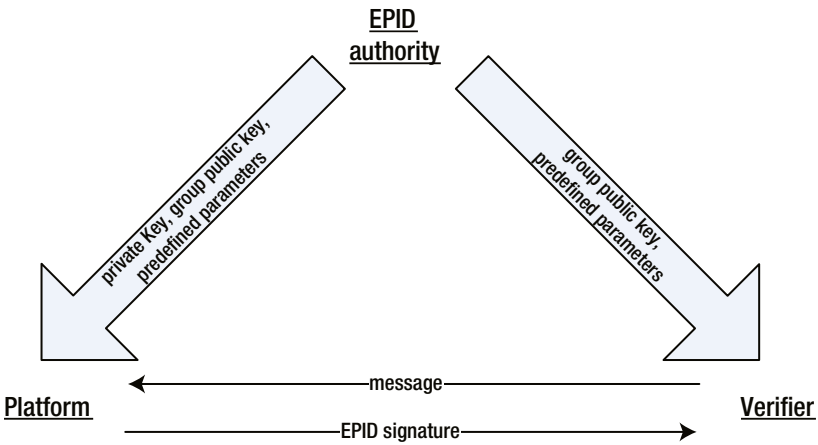

Figure 5-1. *Relationships among the three EPID entities*

During the setup phase, the EPID authority provides private keys for all platforms, respectively. Note that every platform has a different and unique private key. The delivery and storage of the private key must be protected (encrypted) to avoid leakage. The EPID authority also establishes and manages a server for all verifiers to retrieve EPID group public keys and EPID parameters. These materials are not secrets, are digitally signed by the EPID authority, and therefore can be delivered through open networks. Because platforms served by the verifier may belong to different groups, the verifier must obtain all group public keys used by the clients it is serving beforehand from the EPID authority.

In addition to the private key, the platform also needs the corresponding group public key and predefined parameters to generate EPID signatures. The group public key and parameters can be acquired from the EPID authority together with the private key or from a verifier.

Once the verifier and the platform are both provisioned with required information, the platform can sign messages or challenges from the verifier, and the verifier can verify whether the platform's signature is acceptable.

Like all authentication mechanisms, the prover—in this case a platform—must possess a credential and show the verifier that it knows, has, or is the credential. On the other hand, the verifier—in this case the service provider—must have sufficient knowledge beforehand to reliably verify the correctness of the credential presented by the platform.

For example, in the simple password authentication scheme, the credential is the password. The platform must know and present the password. For verification, the verifier must have the expected password or a hash fingerprint of the expected password for comparison.

In the public key authentication scheme, the credential is the private key. The verifier sends a challenge to the platform, and the platform presents a digital signature generated using its private key and the challenge. The verifier must have the platform's public key to verify the validity of the signature.

In two-factor authentication, the second factor is usually "something you have"—for example, a token device with a randomized PIN that is synchronized with the verifier's server. The PSN on a Pentium III processor is also a form of "something you have".

The credential can also be biological or "something you are," such as fingerprint or eyeball characteristics. For example, Apple's iPhone 5s features fingerprint authentication.

None of these authentication schemes is anonymous. The verifier identifies the platform with the presented *unique* credential.

Similar to traditional public key cryptography, an EPID platform owns a unique private key and must keep it secret (protecting with confidentiality). Both the EPID platform and the EPID verifier know the group public key and must maintain its integrity.

Here's what is not so similar to traditional public key cryptography:

- An EPID private key is essentially derived from a random number. The ECC private key is also a random number; but an RSA private key set is not random but a probable prime number. The key generation for EPID is thus faster than RSA.

- An EPID private key has one corresponding EPID public key—the group public key. However, a group public key corresponds to multiple EPID private keys. The number of private keys that map to the same group public key is configurable; it can be as few as several hundred or as many as tens of millions. Obviously, a group with more members, in theory, features better anonymity and offers more privacy. However, as described later in this chapter, under certain circumstances the computational cost will increase linearly with the increase of members in a group.

Intel's security and management engine is the first platform that implements EPID. Intel's chipset series 5 (released in 2008) and newer natively support the EPID platform functionality. A unique private key, in its encrypted form, is burned into security fuses for every chipset part during manufacturing.

For this EPID ecosystem, Intel acts as the EPID authority. Using the private key, the security and management engine proves that it is a genuine Intel platform, and hence eligible for premium services that are only available for Intel platforms.

Key Structures and Provisioning

The platform device must have built-in secure storage capability—at a minimum, encryption, for storing the EPID private key. Integrity and anti-reply protections for the private key storage are optional but recommended. If integrity or anti-reply is absent, attackers may be able to mount denial of service (DoS) attacks against the platform device, so services relying on EPID may not be available when requested.

For EPID version 1.1, a private key is 1312 bits in size. Its components include:

- Group ID (32 bits)
- *A*: An element in a predefined 512-bit elliptic curve group G1 (512 bits)
- *x*: An integer ranging from 0 to $p - 1$ inclusive, where p is the parameter of G1 (256 bits)
- *y*: An integer ranging from 0 to $p - 1$ inclusive (256 bits)
- *f*: An integer ranging from 0 to $p - 1$ inclusive (256 bits)

To save space, an EPID private key can also be expressed in the compressed form, which is 544 bits in size:

- Group ID (32 bits)
- *A.x*: An integer ranging from 0 to $q - 1$ inclusive, where q is the parameter of G1 (256 bits)
- *Seed*: A 256-bit string

The compressed form of a private key must be decompressed before being used for signing. The complete private key can be derived from its compressed form, together with the group's public key and values of all predefined elements such as G1.

On Intel's security and management engine, the EPID key is treated as an asset of the highest value on the engine. On Intel's manufacturing line, an EPID key, in its 544-bit compressed and encrypted form, is retrieved from Intel EPID authority's secure server that generated all keys. The manufacturing process then burns the 544-bit key to secure fuses of the engine. For security and privacy reasons, the key is then immediately and permanently deleted from the secure server. In other words, from that point on, only the part itself knows the value of its EPID private key. As the key is deleted after being burned to fuses, there is no "key retrieval" mechanism. If the part loses its EPID private key, it has to be returned to the Intel factory, and a new EPID private key has to be provisioned to it.

On Intel's security and management engine, because what the fuses store is the compressed form of the private key, EPID is not immediately available for use out of box. To save expensive fuse space, the group public key is not burned to individual platforms. In order to function as a platform, a procedure called "provisioning" must be executed first. The provisioning is a one-time effort for the life of a platform. During provisioning, the group public key and predefined parameters of EPID are sent to the platform from a verifier. The platform uses this data together with its compressed private key to derive the complete private key and then stores the complete private key in secure storage for use in future EPID sessions.

The provisioning must be done before the first invocation of the EPID on a platform. Many platform manufacturers choose to provision in their manufacturing lines for better user experience. Others perform provisioning during system initialization on the first boot. The verifier can be software running on a host operating system or a remote server. Figure 5-2 depicts the provisioning protocol.

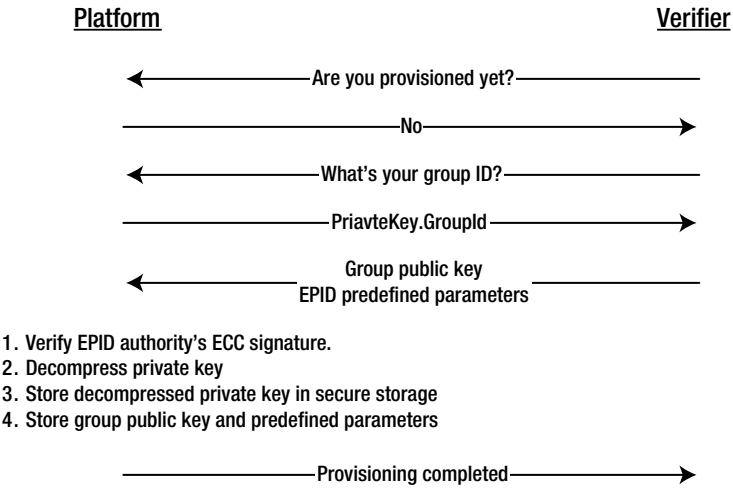

Figure 5-2. *EPID provisioning protocol*

The EPID algorithm uses four mathematical groups: G1, G2, G3, and GT. The groups G1, G2, and G3 are elliptic curve groups. The group GT is a finite field group.

- G1 is 512-bit in size. An element of G1 takes the format of (x, y) where x and y are big integers ranging from 0 to $q - 1$ inclusive.
- G2 is 1536-bit in size. An element of G2 takes the format of $(x[0], x[1], x[2], y[0], y[1], y[2])$, where $x[i]$ and $y[i]$ are big integers ranging from 0 to $q - 1$ inclusive.
- G3 is 512-bit in size. An element of G3 takes the format of (x, y) where x and y are big integers ranging from 0 to $q - 1$ inclusive.
- GT is 1536-bit in size. An element of GT takes the format of $(x[0], x[1], \ldots, x[5])$, where $x[i]$ is a big integer ranging from 0 to $q - 1$ inclusive.

All EPID groups share the same predefined parameters for G1, G2, G3, and GT. These groups are defined by the following parameters:

- Elliptic curve group G1. Parameters:
  - p (256-bit), a prime
  - q (256-bit), a prime
  - h (32-bit), a small integer, also denoted as cofactor
  - a (256-bit), an integer ranging from 0 to q – 1 inclusive
  - b (256-bit), an integer ranging from 0 to q – 1 inclusive
  - $g1$ (512-bit), a generator (an element) of G1
- Elliptic curve group G2. Parameters:
  - p (256-bit), same as in G1
  - q (256-bit), same as in G1
  - a (256-bit), same as in G1
  - b (256-bit), same as in G1
  - coeff (768-bit), the coefficients of an irreducible polynomial
  - coeff[0], coeff[1], coeff[2]: 256-bit integers ranging from 0 to q – 1 inclusive
  - qnr (256-bit), a quadratic nonresidue (an integer ranging from 0 to q – 1 inclusive)
  - orderG2 (768-bit), the total number of points in G2 elliptic curve
  - $g2$ (1536-bit), a generator (an element) of G2
- Elliptic curve group G3. Parameters:
  - p' (256-bit), a prime
  - q' (256-bit), a prime
  - h' (32-bit), a small integer, usually 1, also denoted as cofactor'
  - a' (256-bit), an integer between ranging from 0 to q' – 1 inclusive
  - b' (256-bit), an integer between ranging from 0 to q' – 1 inclusive
  - $g3$ (512-bit), a generator (an element) of G3

- Finite field group GT. Parameters:
  - *q* (256-bit), same as in G1
  - coeff (768-bit), same as in G2
  - *qnr* (256-bit), same as in G2

The public key of an EPID group consists of the following elements:

- Group ID (32 bits)
- *h*1 (512 bits): An element in G1
- *h*2 (512 bits): An element in G1
- *w* (1536 bits): An element in a predefined 1536-bit elliptic curve group G2

Although the group public key and predefined parameters are not secrets, the platform must verify that what is sent by the verifier is trustworthy. The EPID group pubic key and the predefined parameters are digitally signed by the EPID authority using ECDSA.[2] The EPID authority's ECC public key is hardcoded in all platform devices. The platform verifies the EPID authority's ECDSA signature before using the data sent by the verifiers to perform the private key decompression.

If the platform device has enough fuse space, the manufacturing process can provision public key and predefined parameters together with private keys for the devices, in which case the provisioning protocol can be skipped. However, because the provisioning is a one-time procedure for the lifetime of the device, and the public key and predefined parameter are not secrets, it is generally preferable to burn only the private key during manufacturing and perform provisioning before the first use of EPID. This is the design used by Intel's security and management engine to save secure fuse space.

Notice that even if a platform provisions the group public key, predefined parameters, and the private key during manufacturing, it must hardcode the ECDSA root public key of the EPID authority for verifying the verifier's signature in a SIGMA session. See the "SIGMA" section of this chapter for details.

Revocation

The EPID protocol supports revocation of members or groups. How is a platform identified for revocation? In an EPID ecosystem, the EPID authority is the only entity that has the privilege to revoke a member or a group. Once a verifier identifies a platform or group that should be revoked, it notifies the EPID authority with reasons. The authority then examines the request and executes corresponding revocation operations if the request is deemed legitimate. In certain cases, the EPID authority may decide to revoke platforms or groups without requests from verifiers.

Depending on what information is known about the entity to be revoked, a revocation request may belong to any one of three categories: private key-based revocation, signature-based revocation, or group-based revocation. The EPID authority maintains a single group revocation list (GROUP-RL), and for each group that is not on

the group-based revocation list, it maintains a private key-based revocation (PRIV-RL) list and a signature-based revocation list (SIG-RL). The revocation lists are centrally managed and pushed to all verifiers of the ecosystem.

Private Key-Based Revocation

If a member's private key is proved to be possessed by any party other than the platform device itself—for example, if a valid private key is published on the Internet—then the platform device is concluded as having been compromised, and it should not receive any more services as an EPID platform. In order to revoke a member by using its private key, the EPID authority must acquire the value of the private key (in either compressed or complete form). The private key-based revocation can be initiated by a verifier or the EPID authority. Figure 5-3 shows the flows exercised by the EPID authority for revoking a private key.

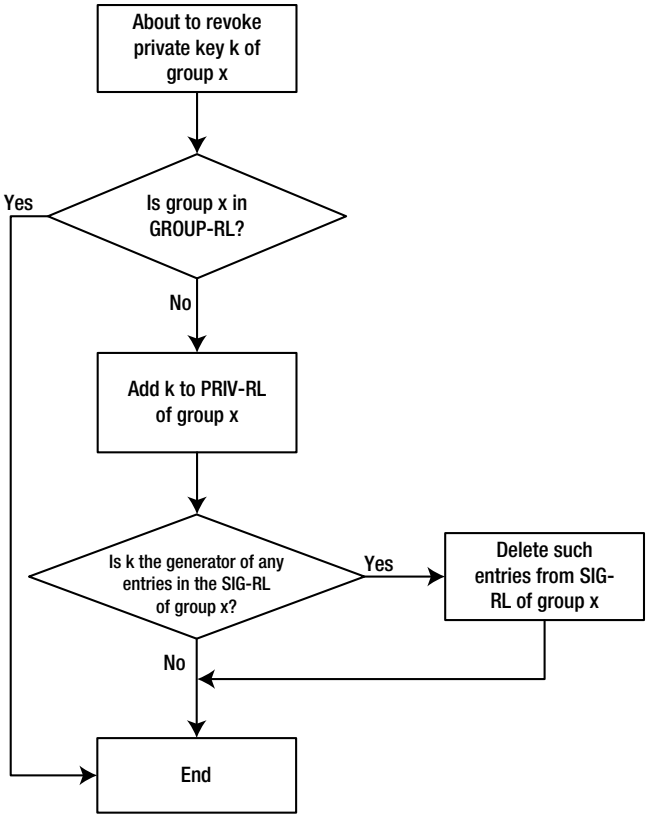

Figure 5-3. *Placing a private key in the PRIV-RL*

Consider a real-world scenario: the owner of a platform breaks into the device by exploiting critical vulnerability of the hardware or firmware and manages to extract the EPID private key from the device. He then shares the private key with other people so they can all enjoy services without having to buy the platform device or pay the service provider. If the provider uses the "base name" option (described later in this chapter), then it can detect such abuse and revoke the private key.

It is possible that the platform to be revoked using private key-based revocation has already been revoked by the signature-based revocation. To minimize the sizes of SIG-RL, before adding a private key to the PRIV-RL, the EPID authority first goes through the SIG-RL and checks if any revoked signatures in the SIG-RL were generated by this key. Such signatures, if any, are removed from the SIG-RL.

Signature-Based Revocation

If a member has reportedly been misbehaving, but its private key is not yet exposed or known by the EPID authority, then the EPID authority may revoke the member by identifying it using a signature the member had previously generated. Misbehaviors are defined by the verifiers and the EPID authority. For example, a platform continuously making excessive requests may be considered to be misbehaving; a platform that repeatedly generates a constant signature for the same challenge is likely compromised, because per the EPID algorithm, multiple signatures generated for the same challenge should be different.

The EPID signature allows the verifier to enforce an optional "based name" parameter so that all signatures generated by the same platform are linkable. The verifier can utilize this option to detect and identify malicious users that abuse anonymity and revoke them using the signature-based revocation mechanism.

The private key-based revocation has higher priority than the signature-based revocation. When the EPID authority receives a signature revocation request from a verifier, it first checks the signature against all entries in the private key-based revocation list. If the signature was generated by a revoked private key, then it will not be placed in the signature-based revocation list.

■ **Note** Although a useful feature, the signature revocation is computationally expensive for both the signature generation of all platforms of this group (even if a platform is not the one that was revoked) and the verifier's signature verification, and it increases protocol traffic between the two parties. Also notice that one revoked member may have more than one signature presented in the signature revocation list.

Figure 5-4 exhibits the flows exercised by the EPID authority for revoking a signature.

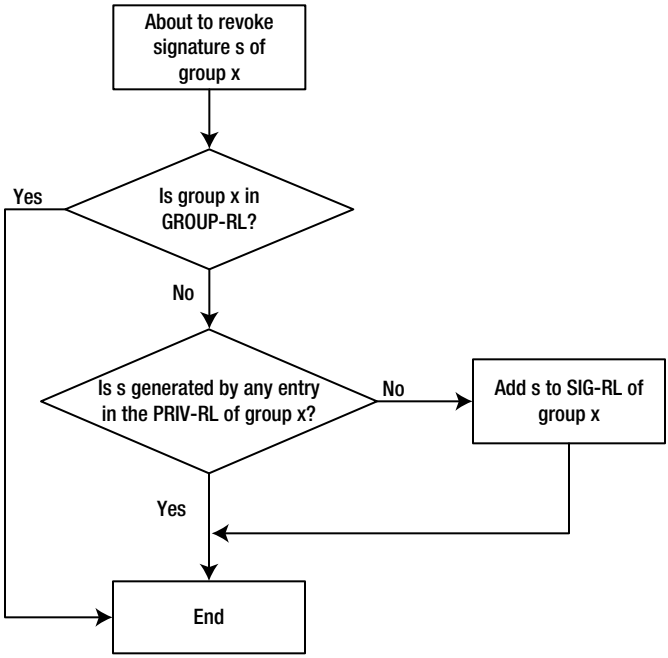

Figure 5-4. *Placing a signature in the SIG-RL*

Group-Based Revocation

This is the mechanism to revoke all members of a group. The reason for revoking the entire group could include termination of the service contract for a group of members, or performance—when too many members of a group have been revoked using signature revocation, the signing and verifying operations can take a very long time. At a certain point, the EPID authority can decide to revoke the group and create a new group.

Another scenario that applies to the group-based revocation is when critical vulnerability in platform implementation is found by the device manufacturer, but the vulnerability has not been exposed or exploited publicly. The vulnerability is critical because it may lead to compromise of the EPID private key. In this case, the platform manufacturer should not only push a patch that fixes the critical vulnerability to all impacted devices, but also revoke these devices using the group-based revocation.

If the group-based revocation is due to performance or platform vulnerability, then the EPID authority will create a new group and reprovision nonrevoked members of the old group. Of course, a member of the old group must show that it has not been revoked by private key and signature, in order to receive a new private key of the new group.

The group-based revocation has highest priority among the three revocation methods. Once a group is revoked, its SIG-RL and PRIV-RL will not be used by verifiers. The EPID authority does not maintain SIG-RL or PRIV-RL for revoked groups.

Figure 5-5 exhibits the flows exercised by the EPID authority for revoking a group.

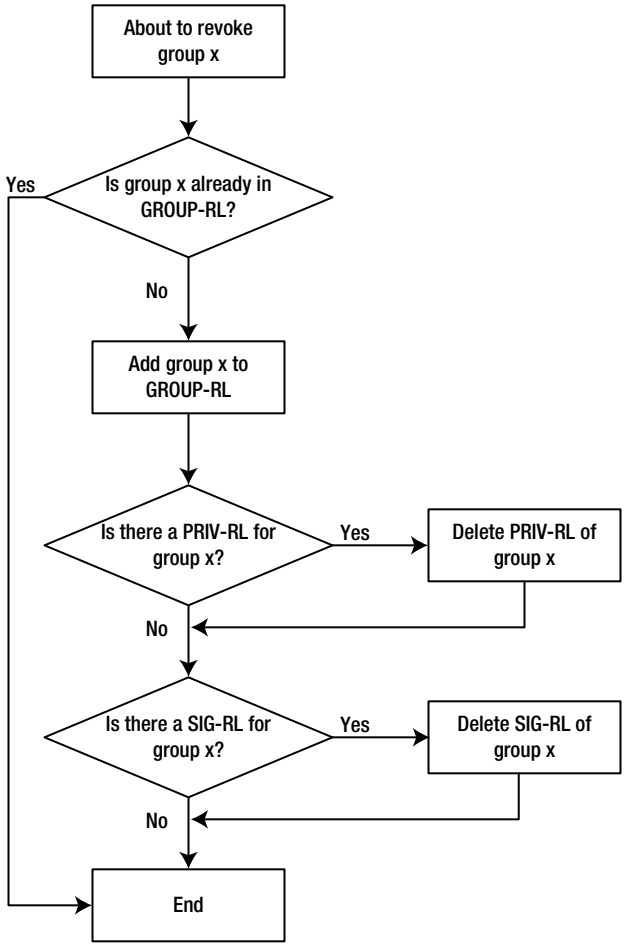

Figure 5-5. *Placing a group in the GROUP-RL*

Signature Generation and Verification

The EPID signature generation is the algorithm implemented by a platform. The EPID signature verification is the algorithm implemented by a verifier. This section gives an interface overview of the two algorithms but does not discuss mathematical details. Readers interested in the detailed steps of the signature generation and verification algorithms should refer to publications listed on the "Reference" section at the end of this chapter.

Figure 5-6 exhibits the communications between EPID entities during an authentication session.

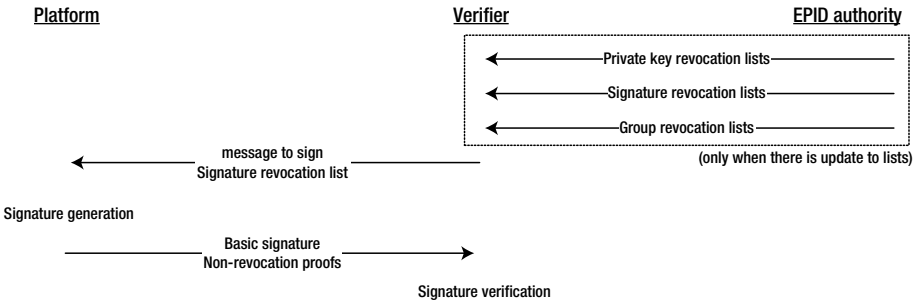

Figure 5-6. *EPID signature generation and verification*

Signature Generation

Input:

- EPID private key
- Corresponding EPID group public key
- Message to be signed
- Verifier's base name (optional)
- Signature revocation list of this group

Output:

- Basic EPID signature
- Non-revocation proofs, one for each entry in the signature revocation list

The basic signature generation is a very intensive operation—it takes as long as two seconds on the security and management engine, which negatively impacts the user's experience. Fortunately, most of the steps of the basic signature generation can be performed without the knowledge of the message to be signed.

The "pregenerating and caching" optimization is utilized widely in the security and management engine by many applications to expedite real-time operation. For the EPID, the engine creates and caches a certain number of "presignatures" without using the message to be signed. The presignatures are stored securely in the engine's internal memory. The presignature generation is performed right after every time the engine is powered on. Alternatively, the unused presignatures in the cache can be stored in nonvolatile memory before power-off, and loaded and reused at the next power-on. Due to the very limited storage space and large size of the presignatures, the first option is deemed more efficient for the engine.

When a signing request is received, the engine fetches a presignature from the cache and completes the final signature using the message from the verifier. The engine then kicks off generation of a new presignature in a low-priority thread, to fill the used slot of

the presignature cache. Experiments show that the presignature generation contributes to about 95% of total basic signature generation time. In other words, the caching optimization reduces the EPID signing time by about 95%.

In addition to the basic EPID signature, the platform has to generate a non-revocation proof for every signature in the SIG-RL. The data transmission and computation time become noticeably long as the number of entries in the SIG-RL increases. Unfortunately, the non-revocation proof cannot be precomputed because the calculation uses the basic EPID signature as input.

If the platform's private key was used to generate at least one signature on the SOG-RL, then the non-revocation proof calculation will fail, and the platform, if not compromised, should abort and notify the verifier of the revocation of itself. Even if the platform is malicious and returns an invalid non-revocation proof for the revoked signature, the verifier will be able to detect it because the signature verification will fail.

Base Name

The base name is an optional input to the signature generation and verification algorithms. It is the verifier that decides whether to require it or not. The platform has the right to deny using a base name. If used, all signatures generated by a platform with the verifier's base name are linkable to the verifier. That is, the verifier is able to identity which signatures come from the same platform. The verifier cannot tell which platform it is, though.

Apparently, this option degrades the privacy protection for the platform. Therefore, the design guideline for the platform is to reject signature requests with base name by default and only use a base name with trusted verifiers. The platform should hardcode a list of trusted verifiers. The SIGMA protocol, introduced in the next section, provides a way to identify the verifier.

But why does a verifier want to use the base name option? The main reason is to prevent rogue platform users from abusing the anonymity of EPID. Consider a scenario where an online movie subscription service sets a limit of 100 movies per month. The base name signature can help the service provider keep track of usage for all platform subscribers, identify excessive usages, and revoke rogue platforms if necessary.

Signature Verification

Input:

- EPID group public key
- Private key revocation list
- Signature revocation list
- Group revocation list
- Message that was signed

- Verifier's base name (optional)
- Basic EPID signature
- Non-revocation proofs

Output:

- Signature valid or invalid

As shown in the input and output parameter lists, the amount of data transmitted between the platform and the verifier for signature revocation is proportional to the number of entries in the signature revocation list. This is because the verifier must send all signatures in the SIG-RL to the platform, and the platform must prove that it is not the generator of any signature in the list.

A lengthy SIG-RL is not only affecting the volume of data transmission, but also increasing the computational cost. The platform must calculate a non-revocation proof for each revoked signature; the verifier must verify the validity of each non-revocation proof sent by the platform.

For Intel's EPID ecosystem, the threshold for the number of entries in the signature revocation list of a group is set to 50 and enforced by Intel's EPID authority. The number was chosen based on performance measurement of the security and management engine, and the capacity and bandwidth of the communication channel between the engine and the verifier host software or the remote verifier server. When the number of revoked signatures exceeds 50, the group will be revoked, and a new group will be assigned to replace the old group.

The verifier must validate the platform against all three revocation lists, in the following order:

- Is the platform's group ID in the group-based revocation list? If so, abort.
- Is the platform's basic signature generated by any private key in the private key-based revocation list? If so, abort.
- Are the platform's non-revocation proofs valid against all entries in the signature-based revocation list? If not, abort. The signature revocation is checked last because its operation is much slower than the other two revocations.

The signature verification is also a mathematically intensive operation. Because the verifier usually is equipped with strong computational capability (faster CPU and more memory), though, it does not introduce significant latency from an end user's perspective.

SIGMA

The EPID provides a solution for a platform to authenticate itself anonymously to a verifier. The EPID is a one-way authentication protocol, because the verifier is not authenticating to the platform. However, for many use cases, the platform has to identify and trust the verifier.

One application of the security and management engine was the Intel Upgrade Service[i] that allowed customers to enable advanced CPU features by purchasing a $50 upgrade card. In this usage model, the engine is the EPID platform, and a remote server set up by Intel is the verifier.

On the one hand, the remote server must be assured that the target platform is indeed an eligible Intel security and management engine. On the other hand, the engine must verify that the upgrade request indeed came from a legitimate Intel server after payment was cleared, and not from a hacker's forgery. Notice that in this case there is no privacy requirement on the verifier. So the authentication method of this direction can be realized by traditional public key cryptography.

In addition to mutual authentication for each other, the platform and the verifier have to perform further secure message exchanges for application-specific protocols. The EPID algorithm per se does not establish session keys for encryption or integrity.

On top of EPID, ECDSA, and the Diffie-Hellman[3] key exchange scheme, Intel designed a protocol called SIGMA (SIGn and Message Authentication) that enables two-way authentication, one direction anonymously and the other distinctively, as well as session key agreement. The security and management engine implements the platform side of the SIGMA protocol.

Verifier's Certificate

The SIGMA protocol is built on a public key infrastructure (PKI). In this PKI, the EPID authority also serves as the root CA (certification authority) of verifiers and issues X.509 certificates to qualified verifier applications. For example, a DAL (dynamic application loader; see Chapter 9) applet that invokes EPID on the security and management engine should obtain its verifier certificate chain rooted to the Intel EPID authority CA. The server for the Intel Upgrade Service obtained its X.509 verifier certificate from the EPID authority as well.

As the root CA, the EPID authority may issue a leaf certificate for a verifier program or issue intermediate CA certificates, which then sign and issue other intermediate CA certificates or verifier certificates. Nevertheless, the verifier's certificate chain must be rooted to the self-signed certificate of the EPID authority. Recall that the platform device with compressed private key must hardcode the EPID authority's root ECDSA public key for verifying the EPID group public key and predefined EPID parameters sent from the verifier. The same ECDSA public key is also used by the platforms to validate the certificate chain of the verifier.

The SIGMA PKI supports revocation of the verifier. When a verifier is no longer qualified as an EPID verifier, and its X.509 certificate has not yet expired, then its certificate can be revoked by the EPID authority. The criteria of a qualified verifier are defined by and at the discretion of the EPID authority. For example, upon end of life, the certificate of Intel Upgrade Service was revoked.

One or more online certificate status protocol (OCSP) servers are employed to enforce the revocation. The EPID authority issues X.509 certificates to OCSP servers. The EPID authority pushes the revocation lists of intermediate CAs and/or verifiers to all OCSP servers whenever new certificates are revoked.

---

[i]Intel Upgrade Service was end-of-life in 2011 and no longer available to customers.

Upon request, the authorized OCSP servers issue signed non-revocation proofs for all intermediate CA certificates and verifier certificates to the requesting verifiers. The non-revocation proofs include timestamps that will be used by the platform. The verifier can then present the non-revocation proofs together with its certificate chain to the EPID platform.

As an embedded system, the security and management engine does not have convenient network access. Therefore, the SIGMA protocol is designed such that the platform does not communicate with the OCSP server directly but only connects with the verifier.

Let's summarize all materials signed by the EPID authority's root ECDSA key:

- EPID group certificates that contain group public keys
- EPID predefined parameters
- Verifier's certificates and intermediate CAs' certificates
- OCSP servers' certificates

Messages Breakdown

A high-level overview of the SIGMA protocol is given in Figure 5-7. Detailed descriptions follow.

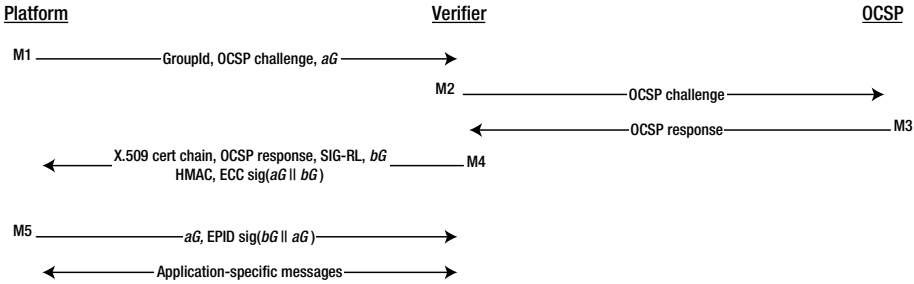

Figure 5-7. *SIGMA protocol*

To begin a SIGMA session, the platform randomly generates an elliptic curve Diffie-Hellman private key a and calculates public key $a \cdot G$. The base point G is predefined by the EPID authority. The verifier similarly generates b and calculates $b \cdot G$.

In M1, the platform sends its EPID group ID and Diffie-Hellman pubic key $a \cdot G$ to the verifier. The group ID is for the verifier to look up and send back corresponding SIG-RL for that group in M4.

Under certain cases, the platform can also send a random OCSP challenge in M1, if it wants to receive a real-time "noncached" OCSP response (non-revocation proof). If an OCSP challenge is not sent, then the verifier is allowed to provide a "cached" OCSP response that was previously generated by the OCSP server. It is up to the platform implementation to decide the maximum age of a cached OCSP response that is considered acceptable. The security and management engine accepts an OCSP response that was generated within the last 24 hours. In other words, the verifier program may

vulnerably retrieve a non-revocation proof from the OCSP server every 24 hours, for example at midnight. In the case of a cached OCSP response, the challenge is not material and will not be checked by the platform.

Obviously, if the platform decides to request for a noncached OCSP response, the SIGMA session will take significantly longer because the platform has to communicate with the OCSP server via the verifier during the SIGMA session (messages M2 and M3). If the platform accepts cached OCSP response, then M2 and M3 can be skipped. Another more problematic scenario with noncached OCSP response is when the OCSP server is temporally unavailable or unreachable by the verifier, in which case the SIGMA session has to be aborted, resulting in a poor user experience.

So under what conditions should a platform request a noncached OCSP response? The answer is application specific. Intel's security and management engine requests for noncached OCSP responses only when

- *The engine has not been provisioned date/time yet.* As a platform, the engine must have the current date/time to confirm validity periods of verifiers' X.509 certificates and other PKI elements. If the engine has no date/time information, then it has to ask for a noncached OCSP response and use the timestamp in the OCSP response as the current date/time. The engine's kernel has a secure clocking capability (refer to Chapter 3 for details) and will maintain the trusted date/time, even if the device is powered off.
- *The date/time was provisioned more than 30 days ago.* Every 30 days, the engine requests for a noncached OCSP response to calibrate its date/time. This eliminates the influences of possible glitches of the internal clock.

M4 is a heavily loaded message that deserves more attention. M4 contains the following:

- *M4.1*: Verifier's Diffie-Hellman public key $b{\cdot}G$
- *M4.2*: SIG-RL
- *M4.3*: Verifier's X.509 certificate chain
- *M4.4*: OCSP response
- *M4.5*: HMAC on M4.1 to M4.4
- *M4.6*: Verifier's ECDSA signature on $a{\cdot}G \,||\, b{\cdot}G$

Before sending M4, the verifier uses its ECC private key to sign "$a{\cdot}G$ concatenated with b·G." It also derives the Diffie-Hellman shared secret s from b and $a{\cdot}G$. The HMAC on M4.1 to M4.4 are calculated using s.

The platform verifies validity of the certificate chain and the OCSP response, including checking the nonce if it is noncached, and then verifies the ECC signature. The platform then calculates the Diffie-Hellman shared secret s from a and $b{\cdot}G$ and verifies the HMAC. Once everything checks out, the platform proceeds with EPID signature generation on message $b{\cdot}G \,||\, a{\cdot}G$, and sends to the verifier in M5 with $a{\cdot}G$. $a{\cdot}G$ is sent again in M5, so the verifier is able to match the $a{\cdot}G$ values in M1 and M5 and confirm they belong to the same SIGMA session, when there are multiple concurrent SIGMA sessions.

After the verifier verifies the platform's EPID signature (including non-revocation proofs) that has been sent in M5, the two parties have completed mutual authentication and session key agreement. The subsequent messages between the platform and the verifier are application specific. Derived values from the shared secret *s* are used as an encryption key and an HMAC key, respectively, to safeguard the application-specific communication.

The lifetime of a SIGMA session is determined by the platform and the verifier. Though a maximum lifetime does not have to be enforced, it is recommended that a new SIGMA session be established periodically. For performance considerations, a SIGMA session should not be renewed too frequently, because EPID is a relatively slow algorithm.

Implementation of EPID

This section discusses the best-known methods for implementing EPID infrastructure.

Key Recovery

Due to its nature of anonymity, the EPID must be a native built-in functionality of the device. The EPID private keys should be provisioned to platform devices during manufacturing, instead of in the field. This is because as soon as a consumer purchases the device, the device is associated to the consumer and is no longer anonymous. Any key provisioning in the field would have to involve the device owner's identity and actions; for example, using a credit card to purchase an EPID-based service.

Reprovisioning an EPID private key requires a "super verifier" that has access to the EPID key generation server; hence the super verifier must be set up by the EPID authority. During a SIGMA session, if a verifier finds that the platform's group is revoked, then it should direct the platform to the super verifier and the platform should check whether EPID reprovisioning is available for this platform.

As briefly discussed earlier, Intel's security and management engine supports reprovision of EPID private key in the field for one of the following two reasons:

- Performance due to too many signatures revoked by the signature-based revocation
- Critical vulnerability in firmware

In the first case, the platform presents a signature generated using its current EPID private key in a SIGMA session with the super verifier. Once the SIGMA session is established, the super verifier will send the new group ID and a complete private key to the platform. The platform then replaces the old key with the new key in its secure nonvolatile storage.

The second case is more complicated. If vulnerability is found in firmware that may leak the EPID private key, and there is no known exploit against the vulnerability yet, then the manufacturer should release a firmware hotfix and push the firmware update to all platforms that have the vulnerability. In the SIGMA session (established using the old private key from the platform's perspective), the super verifier must first confirm

the platform has the latest firmware installed, and then send a new private key to the platform. From then on, the old private key is obsoleted, and the platform must use the new EPID key in all future SIGMA sessions.

■ **Note** The firmware hotfix must also replace the secret keys that are used to protect nonvolatile files.

It is always tricky and costly to deal with consequences of critical bugs found in released products. For some cases, a recovery may not be an ideal solution. For example, if there were already published exploits against the vulnerability, then those rogue end users that had exercised the exploit and retrieved the private keys would be eligible for reprovision with a new private key and continue to enjoy premium services that they were not supposed to receive.

If the vulnerability also allowed compromised firmware to cheat the super verifier by sending an arbitrary firmware version number in the SIGMA session, then the super verifier would happily send the new private key to the vulnerable firmware.

Attack Mitigation

Like all cryptography protocols, attackers target two aspects: algorithm and implementation. To date, there are no known attacks against the EPID algorithm.

To protect an asset, the requirement of the implementation is to make the cost of successful attacks higher than the value of the asset. If an asset can be compromised on one device and used on all devices (BORE attack), then it is a high-value asset and must be afforded the strongest protection. For example, global keys that are stored in all devices fall into this category. On the other hand, device-specific secrets are of less value than global secrets. The attack must be repeated on an individual device to retrieve the secret from the device. This is impractical, especially if the attack requires special hardware, setup, and expertise to mount.

The EPID private key is a highly valuable asset because it grants access to premium services offered exclusively to the platform. The implementation of the security and management engine attempts to ensure that the EPID key cannot be revealed from a device without special and expensive equipment and advanced expertise in hacking. The following mechanisms to protect EPID private keys are applied:

- The EPID keys in fuses are in encrypted form.
- Anti-cloning: The decompressed private key is stored in secure nonvolatile storage and protected with AES for encryption and HMAC-SHA-256 for integrity. The AES and HMAC keys are unique per part. Therefore, copying the EPID key file on the flash from one part to another will not work.
- At runtime, the EPID key is handled in the engine's internal memory and is never exposed in the clear to the host.

Applications of EPID

The services and applications that are built on the EPID always have dependency on certain features of the device. Intel's security and management engine features premium applications that are only available on the engine and should not be executed by other products. These features require specific hardware and/or environment support to function. The EPID is used to prove it is *a* genuine Intel platform with such support and is eligible to enjoy the premium services, but not *which* individual Intel platform.

Examples of such premium services include:

- *Intel upgrade service*: Consumers could purchase an upgrade code from Intel and unlock advanced CPU features. This was the first application of EPID, which was dropped in 2011.
- *Anti-theft technology*: Shut down a mobile device when the owner reports it as stolen.
- *Premium content playback*: Intel platforms feature proprietary PAVP (protected audio video path) technology that offers robust hardware protection for contents (see Chapter 8 for details). Once a platform is authenticated via EPID to be a genuine Intel platform, the user can enjoy premium contents (such as high-definition movies) that are only allowed, as required by the content creators, on platforms with hardware-level content protection.
- *Identity protection technology*: Intel's identity protection technology[4] provides a simple and tamper-resistant method for protecting access to customer and business data from threats and fraud. EPID is used to authenticate the Intel platform.

Next Generation of EPID

Intel continues working on improving the EPID scheme and exploring new deployments for the EPID.

Two-way EPID

The two ways in SIGMA's two-way authentication are not equal—one direction is anonymous, and the other is not. As more applications deploy EPID, it is likely that for some applications, both sides of the protocol must remain anonymous. In that case, both parties will implement platform and verifier functionalities, and two EPID sessions must be established to realize mutual anonymous authentication.

Optimization

Intel's chipset series 5, 6, 7, and 8 and Bay Trail systems-on-chip products feature the second version of EPID: EPID 1.1. The sizes of its elements (keys, parameters, signature, revocation lists, and so forth) are not small. The algorithms require a relatively large amount of computational resources.

Performance, memory consumption, storage space, and power consumption are all critical measures for mobile devices. The new EPID 2.0 standard strives to reduce computational cost by choosing more efficient curves and reducing key sizes while maintaining the same security level. The EPID 2.0 is published as ISO/IEC standard 20008.[5]

References

1. Brickell, Ernie, and Jiangtao Li, "Enhanced Privacy ID: A Direct Anonymous Attestation Scheme with Enhanced Revocation Capabilities," *IEEE Transactions on Dependable and Secure Computing*, May/June 2012, pp. 345–360.
2. National Institute of Standards and Technology, Digital Signature Standard (DSS), http://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-4.pdf, accessed on November 10, 2013.
3. Network Working Group, "Diffie-Hellman Key Agreement Method," http://tools.ietf.org/html/rfc2631, accessed on November 10, 2013.
4. Identity Protection Technology, http://ipt.intel.com/, accessed on April 20, 2014.
5. ISO/IEC JTC 1, "Anonymous digital signatures", ISO/IEC 20008-2, 2013.