Supporting Flexible Streaming Media Protection through Privacy-aware Secure Processors

Youtao Zhang [a,*], Jun Yang [b], Lan Gao [c]

[a] *Computer Science Department, University of Pittsburgh Pittsburgh, PA 15260*

[b] *Electrical and Computer Engineering University of Pittsburgh Pittsburgh, PA 15261*

[c] *Computer Science and Engineering Department, University of California at Riverside, Riverside, CA 92721*

Abstract

Due to the explosion of Internet technology in the last decade, there is an increasing demand for secure and effective streaming media protection (SMP) in the new computing environment. Since end users usually have the full control of their machines, pure software based approaches such as user/password validation and group key based content encryption, are not sufficient to defend many attacks, in particular, malicious key sharing. On the other hand, existing hardware-based approaches tend to be too restrictive to adopt.

The emerging secure processor designs provide a new direction for hardware assisted streaming media protection (H-SMP). The research in the computer architecture community has shown that secure processors can help to defend various types of attacks such as those with a hijacked and malicious OS. However existing designs focus on securing point-to-point data transfers and face both privacy and performance issues when supporting group-oriented applications e.g. video on-demand. In this paper, we present privacy-aware secure processor designs for H-SMP against key sharing. We first categorize different protection policies, compare their advantages and disadvantages, and then discuss the novel hardware enhancements including instruction set extensions for supporting these policies. We elaborate the implementation details and present the security and performance analyses.

Key words: Digital Content Protection, Secure Processor, Group Key Management

* Corresponding author.
Email address: zhangyt@cs.pitt.edu (Youtao Zhang).

1 Introduction

The explosion of Internet technology in the last decade has fertilized the enormous growth of novel applications such as video on-demand and IP-TV. While these applications provide customized content delivery and better interactivity, a major challenge to the related industry is how to achieve effective protection against various types of attacks. Losses of billions of dollars each year have been reported due to insufficient protection in the new computing environment [1].

Protecting the streaming media delivery over the Internet differs in many ways from that in traditional settings such as cable-TV. For example, a cable company usually installs a set-top box for each of its clients. The set-top box includes necessary secret data and hardware to descramble analogy cable channels and decode encrypted digital contents before feeding them to the TV set. In this setting the content provider (the cable company) and the consumer tend to be coupled for several months or more, which helps to amortize the cost to install and reclaim set-top boxes. However the content provider/consumer relationship in Internet-based applications tends to much looser — a web user may choose to watch a football game from one server but never return. Asking users to buy and install customized set-top boxes from all potential content providers is clearly infeasible.

While approaches have been developed to protect digital rights over the Internet, most of them are software based and insufficient to defend wide range of Internet-based attacks. For example the widely used user/password mechanism can only provide limited protection. The password may be hacked and/or multiple users may share one password. In addition, the password is often used as entry validation but not to encrypt the contents. For broadcasting oriented applications, encrypting the contents differently for each client can easily overload the server. Group key management schemes [13] help to reduce this overhead by sharing the key within the group. Clearly the group key needs to be changed frequently to ensure the proper delivery of contents to the desired user group. While different group key management schemes have been proposed, they are still insufficient to defend attacks such as key sharing where illegal users get the user/password pair or the initial video decryption key (i.e. session key) from a (hacked) current user, and then mimic the behavior of the legal user including receiving services, updating session keys, etc.

The hardware-assisted techniques for digital right protection are emerging recently. However many are too restrictive for both users and content providers. For example, the DTCP approach [2] requires all related devices such as the computer, the TV-set, the display, are manufactured under the DTCP-license and subject to fee charges, which restricts its wide adoption.

On the other hand, the pervasive security requests of modern applications have advocated the design of secure processors [4,14,17] which integrate specially designed cryptographic units on the processor chip. These designs can provide high level security protection e.g. even the OS and the other hardware components are hijacked as well. This is important as in many cases, the end users are malicious and may choose to hack their hardware to break the security protection. As an example, one way to hack SONY playstation is to purchase and mount a modchip to the playstation hardware. The playstation can then play any game CDs without copyright validation [5].

A secure processor design can provide high level security protection without introducing too much restriction to normal users. However the current designs are not effective for streaming media protection over the Internet. A secure processor usually embeds a unique private/public key pair to secure across-processor information exchange. It lacks sufficient support for broadcasting oriented applications. The per-processor unique key pair virtually identifies the processor and the user over the Internet and can introduce privacy problems [18]. A similar example is that, Intel included unique sequence number for Pentium III processors but had to disable it due to widespread privacy concerns [3].

In this paper we propose privacy-aware secure processor designs to support flexible streaming media protection (SMP). We first categorize the hardware-assisted SMP (H-SMP) according to (i) if the video decryption session key is visible to end users and (ii) if the server needs to generate different key update packets to update a session key. In general, a scheme provides better security if the session key is invisible to the end user while it provides better performance if few key update packets are generated. However, the former is at the cost of less compatibility and less adaptability while the latter tends to require more hardware. We propose a privacy-aware secure processor design with carefully calibrated ISA (instruction set architecture) extensions to support these policies. We elaborate the implementation of these policies and analyze their security and performance.

For the rest of the paper, section 2 presents the four hardware assisted policies against malicious key sharing. We present the privacy-aware secure processor design in section 3, the policy implementation details with security analysis in section 4, and the performance analysis in section 5. Section 6 concludes the paper.

2 HSMP: Hardware-assisted Streaming Media Protection

Let us first look at a video on-demand service example (Figure 1) in which the server broadcasts the video to users over the Internet while users may interactively communicate with the server for different choices. To protect digital contents with sufficient strength but modest cryptographic computation overhead, the server uses user/password to admit legal users and then encrypts the contents using a dynamically changing *session* key that is known to these users who form a dynamic group. A group member, when he wants to view the contents, needs to fetch the current session key e.g. key1, and then decrypt the encrypted contents accordingly. Here we do not limit the receiving of encrypted video contents by illegal users as it is inherently difficult to prevent over the Internet. Contents may be broadcasted over the wireless LAN, and a proxy server or a router can cache its received contents.

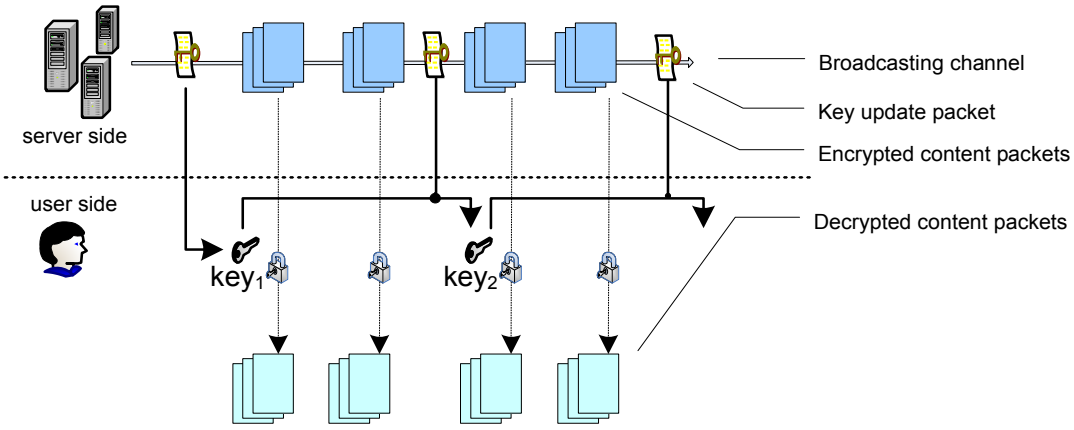

Fig. 1. The encrypted content broadcasting setting.

Due to the loose content provider/consumer relationship, there might be a need for fine-granular billing plan i.e. a user should not be charged for a month, but only for the exact amount of time (at minute or second level) that he uses the service. The charging ends whenever he decides to stop and leave. With this requirement, the session key should be changed periodically such that a different key is used after a new user joins or an existing user leaves. A user cannot view the contents before his join or after his leave. Accordingly the server sends out key update packets periodically to update keys saved on legal users, such as the key update from key1 to key2 in the figure.

Pure software based SMP schemes are insufficient. Since an attacker has the full control of his machine, assume he gets the initial session key, he can mimic all actions that a legal user does in key updating. It is then difficult to exclude him from the group. The initial session key may be obtained by hacking a user/password pair or receiving the key directly from a malicious key sharer. There are heuristic defending schemes e.g. some news subscription

services [16] restrict the number of IP addresses that users may use to read the subscribed newsletter. However it is not hard to bypass using a web proxy which hides users' IP addresses. Malicious key sharing is a serious threat to software based SMP since the required hacking effort is minimal — except getting the user/password pair or one session key from the key sharer, illegal users do not ask for any additional help from the key sharer.

Existing hardware assisted approaches tend to be restrictive. A cable-TV-like approach would request content providers to deploy customized *set-top* devices to Internet users; a DTCP-based protection would request all devices be manufactured under the DTCP license agreements and subject to charges [2].

2.1 Hardware assisted SMP (HSMP) policies

In this section we discuss how secure processors can help streaming media protection. We start from the simple design as those in [4,14,17], a secure processor has a unique private/public key pair per processor a.k.a. PKI (public key infrastructure). While the public key is well-known, the corresponding private key is only known the processor, i.e. it is *invisible* to the OS, the application, and even the owner of the machine.

As shown in Figure 2, a secure processor supports secure sensitive information exchange among different processors (and their users). For example, to update a session key, the server can encrypt the new key using the public key of the secure processor such that only the corresponding processor (a unique one) can process the packet and get the new key. Illegal users cannot get their new keys even if they can receive the key update packets and have the current and old keys.

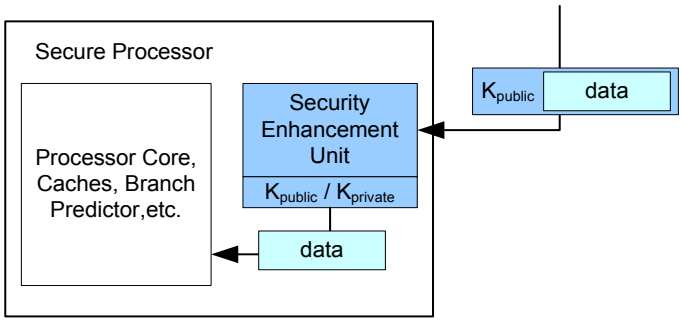

Fig. 2. A simple secure processor design.

The design goal of hardware assisted streaming media protection (HSMP) using secure processors is to secure the session key update such that it is significantly more difficult to receive streaming service illegally. HSMP is not

designed to completely terminate piracy which is almost impossible over the Internet. In an extreme case, a determined hacker can record a movie using his video camera when the movie is played on the TV. In a PC setting, unless the media player software is integrated on the hardware, and/or the display hardware is specially manufactured, the attacker can intercept the streaming packets to reconstruct a copy. As we discussed requesting too many secure devices significantly restricts its wide adoption. In practice the above attack is less popular as (i) it does not allow interactions with the server; and (ii) the quality of the pirated version tends to be low due to possible connection congestion, and hardware setting difference etc.

Using a secure processor enabled HSMP, we restrict illegal users from processing their received packets at different levels. In this section we categorize four different HSMP polices according to two criteria: (i) if the session key is known to the end user; and (ii) if the server needs to generate different key update packets for different users. We next discuss the advantages and disadvantages of each scheme.

		If the same key update packet is used	
		Different	Same
If the session key is visible to the user	visible	HSMP-1	HSMP-2
	invisible	HSMP-3	HSMP-4

Fig. 3. Categorizing different HSMP polices.

- HSMP-1: this is a naive adoption of secure processor designs to support SMP. The server encrypts a session key using the public key of a secure processor. Since each secure processor has a unique public/private key pair, the transmitted key update packets are different for different users. At the user side, the session key is visible to the end user as well as the video processing software such that it can be used by different decryption and video playing software. Therefore it is less restrictive.
- HSMP-2: For each key update in HSMP-1, the server has to generate multiple different key update packets using expensive public key cryptography. The computation could become a performance bottleneck when the number of users is large. For this reason, we design HSMP-2 in which the server just generates O(1) packets to update all users. This packet is broadcasted to all users and with additional hardware support, and enables the key update. The tradeoff between HSMP-1 and HSMP-2 policies is that HSMP-2 requires more hardware than HSMP-1. We will elaborate the details in following sections.

  Since the session key is visible in both HSMP-1 and HSMP-2, a determined malicious key sharer can still share his key with many illegal users. It

is more tedious since the hacker has to extract and redistribute the sequence of session keys synchronously and timely. If the key is frequently updated, the hacker's computer has to stay online to "serve" illegal users, which makes it easier to track and locate the attacker. In summary, HSMP-1 and HSMP-2 tend to overburden the hacker and thus discourage key sharing.

- HSMP-3 and HSMP-4: To further enhance digital contents protection, we design HSMP-3 and HSMP-4 which hide the session key by storing it in a tamper-resistant register. This register can only be accessed by specially designed onchip security enhancement hardware components. That is, an onchip decryption unit is responsible for decrypting received video packets. Its output is the plaintext to be fed into video player software. Since the end user cannot get the plaintext session key, key sharing is no longer possible. An attacker may choose to share the contents directly by receiving encrypted video packets, decrypting them, and rebroadcasting them to illegal users. We believe that the nontrivial bandwidth and other requirements for this hacking practice defeat most attackers.

  The difference between HSMP-3 and HSMP-4 is the number of needed packets for each key update. Similar as HSMP-1 and HSMP-2, the server generates different packets in HSMP-3 but O(1) packets in HSMP-4.

3 Privacy-aware Secure Processor

In this section we first present our privacy-aware secure processor design. We discuss several design issues to clarify why the proposed components in our design are necessary. We then present four new instructions for accessing the proposed hardware components.

3.1 The architectural design

Figure 4 illustrates the architectural components of our privacy-aware secure processor design. It consists of seven secure registers and a security management function unit. All components are integrated onchip and accessible through special instructions.

- Two tamper resistant registers for storing the private/public key pair of the secure processor. HSMP adopts 256-bit ECC-based PKI public key cryptography i.e. both registers are 256-bit long. The private key is kept secret while the public key is known to the public. Instead of assigning a different PKI pair for each processor, we allow the same PKI pair be used for a batch of processors.

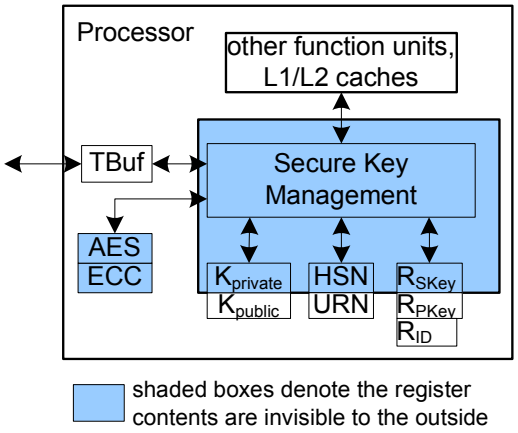

Fig. 4. The privacy-aware secure processor design.

- A tamper resistant register for storing the hardware sequence number (HSN), a per-processor based 128-bit value. We assign a unique HSN for each processor in a processor batch. HSN register is *invisible* to the public but accessible through the security management unit. Together with a user provided random number (URN), HSMP can provide privacy-aware security management.
- Three tamper resistant registers: a 127-bit register (R_{ID}) for storing an application level identifier, two 128-bit registers (R_{skey} and R_{pkey}) for storing a secret (user invisible) session key, and a public (user visible) session key respectively. While R_{pkey} is visible to the user, it can only be overwritten through the onchip security management unit.
- A 512-bit data buffer `TBuf` for security management. It is designed to buffer data before writing back to the memory or after loading from the memory. It is not tamper-resistant.
- The hardware assisted SMP (HSMP) security management logic. It is integrated onchip for managing the above registers and support flexible different SMP policies.

3.2 Design issues

In this section, we discuss several issues to elaborate the design philosophy of our privacy-aware secure processor. For each issue, we describe the problem and briefly discuss our solution. We will elaborate the technical details and security analysis in the implementation of HSMP policies.

Privacy protection. As we discussed, the current secure processor design embeds a unique private/public key pair per processor. This may become a serious privacy concern over the Internet.

For example in Figure 5 an end user U purchases the video service online

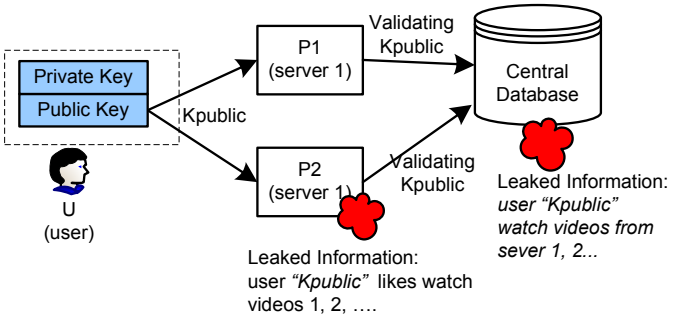

Fig. 5. Possible Privacy Leakage.

from a content provider $P1$. With the current secure processor design, to get the session key, U needs to send his public key K_{public} to $P1$ who will return the session key information in an encrypted packet to be processed by U's processor. For security reasons, $P1$ should be able to attest if the given K_{public} is valid. On the other hand, a normal user U may not want $P1$ to collect his private information by remembering his K_{public}. More elaborately:

(1) The content provider $P1$ should not trust easily the public key passed by U since it could be faked by U in which case the private key is also known to U. If $P1$ uses K_{public} without validation, then U can manually extract any secrecy included in the packet even if the server does not want U to see e.g. the secret session key in HSMP-3. For this reason, $P1$ has to validate the received public key, i.e. if it is the public key of a secure processor manufactured by a trustworthy company. A possible solution is, $P1$ queries a central database created by the manufacturer (Figure 5). However, it is not desirable since the processor manufacturer can then collect and data mine these queries, and extract sensitive information it is not supposed to get.

(2) As the end user has to send his public key in order to receive the session key, a malicious $P1$ can easily identify a returned user if the public key was used before. After some time, the user activities can be collected and analyzed by $P1$ without U's awareness. Furthermore, $P1$ can trade U's information with other sales companies to get a more comprehensive picture of U. This is certainly not what U or any normal customer desires.

The user privacy is violated in both cases mainly due to the fact that a secure processor can be uniquely identified by its public key. This privacy concern is very similar to the serial number that Intel embedded in Pentium III processors [3]. Intel tried to include a unique hardware serial number that can be queried by some software. Due to widespread privacy concerns, the number was *disabled* in the default settings.

To enforce privacy protection, our design replaces the unique per-processor PKI key pair with a combination of three items: (i) a batch processor PKI key pair for security protection e.g. all processors manufactured by a company in

September 2006 share the same pair. The public key is well-known but the secret key is kept secret in an onchip tamper-resistant register. (ii) A hardware sequence number (HSN) that is also kept secret and stored in tamper-resistant register. Each processor in the batch has a unique HSN to distinguish itself from others. (iii) A user controllable random number (URN). With this design, the processor identity is represented by an PKI encrypted block of HSN and URN. While the uniqueness of each processor is achieved with HSN, its identity is obfuscated using URN. The security is ensured using PKI cryptography. Therefore we achieved balance among unique processor identity, security and privacy. We will elaborate the details in the implementation of different HSMP policies.

Asymmetric and symmetric cryptography. With the embedded PKI key pair, a secure processor can perform secure asymmetric cryptography whose key distribution tends to be easier. For example, the public keys of different processor batches may be made well-known without security concerns. However asymmetric cryptography is more expensive. For applications that require frequently session key updates, the server side crypto-computation could become a performance bottleneck even with hardware assistance. In fact, asymmetric cryptography is rarely used for encrypting batch of data but often used for setting up the session key i.e. the initial session key is asymmetrically encrypted and convoyed to the desired receiver while later update can be conducted using the symmetric cryptography.

For this reason, we introduce three secure registers for implementing more efficient symmetric cryptography based key update. We introduce an application level user identifier R_{ID}, a public session key R_{Pkey} and a secret session R_{Skey}. While R_{ID} and R_{Pkey} are known to the end user, the content of R_{Skey} is kept secret and invisible. Depending on the policy to be used, either R_{Skey} or R_{Pkey} may be used as the key to decrypt the encrypted video.

Hardware assisted access control. In most cases, a key update request arises when there is a change in the current group of end users e.g. a user leaves the group. Since a leaving membership may continue receiving the encrypted packets from the broadcasting channel, we need a secure hardware assisted mechanism to prevent it from getting the new session key. The similar need arises if we want to convoy the key to the processor of a new member.

Figure 6 illustrates two types of hardware assisted access control modes for selective key update. Assume each processor has an ID and a secret key. Proc_1 wants to transfer a piece of sensitive information (e.g. a new session key) to a subset of processors,

- In case (a), proc_2 is a new user that just joined the group while proc_3 to proc_{10} are not legal users. The packet containing the new key information

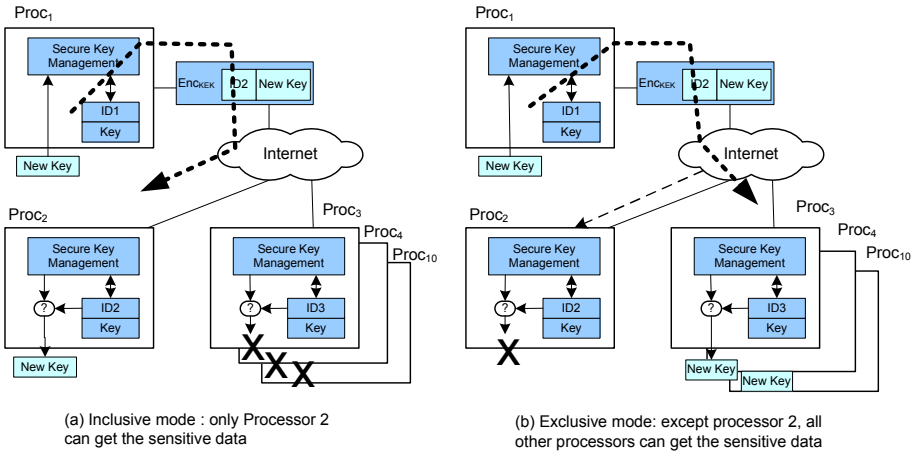

Fig. 6. Modes of Hardware Assisted Access Control.

should only be processed by the corresponding processor. Even though other processors may receive the key update packet, they should not be able to handle it.

We introduce an *inclusive delivery* mode in which each receiving processor checks to see if its ID matches the one in the encrypted block. The sensitive information is extracted only if a match is found.

- In case (b), $proc_2$ is a leaving member while $proc_3$ to $proc_{10}$ are remaining legal users. The packet containing the new key information should only be processed by all processors except $proc_2$.

  We therefore introduce an *exclusive delivery* mode in which a receiving processor extracts and processes the sensitive data in the input block only if a mismatch is found.

The selection of two different data delivery modes is controlled by the server and is encoded in the transmitted message. We will elaborate more in the next section.

3.3 Temper resistant ISA extensions

To exploit the HSMP unit, we introduce four new instructions as follows.

HSMP token preparation. Given a user selected random number (URN) stored in register R1, the secure processor encrypts the concatenation of URN and HSN and stored the encrypted data block in `TBuf`.

HSMP session key initialization. The HSMP unit loads a 512-bit data block from the memory and decrypts it using the private key of the processor. If the message HSN field matches the processor HSN, then a 127-bit identifier is loaded into R_{ID} and the session key registers are initialized. If the access control bit in the message (the 384-th bit) is cleared, then the secret session key

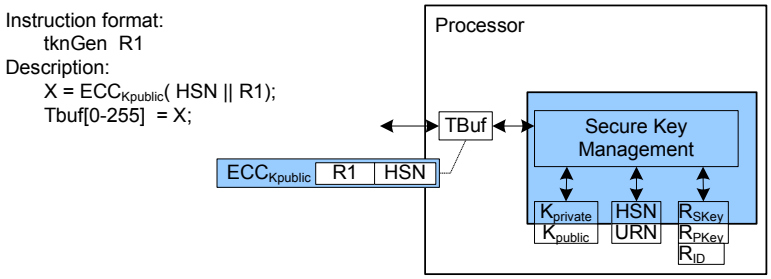

Fig. 7. The logic of instruction `tknGen`.

register R_{Skey} gets the key from the message and R_{Pkey} is cleared. Otherwise R_{Pkey} takes the key value and R_{Skey} is cleared.

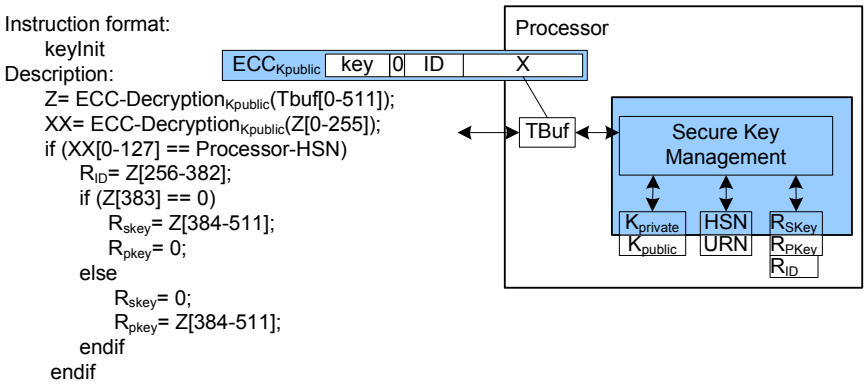

Fig. 8. The logic of instruction `keyInit`.

HSMP session key reset. The HSMP unit decrypts a 256-bit encrypted data block from `TBuf`. If its first 127 bits match the onchip ID, then the key value in the message is used as the current session key. The 128-th bit in the message guides which session key register should be overwritten

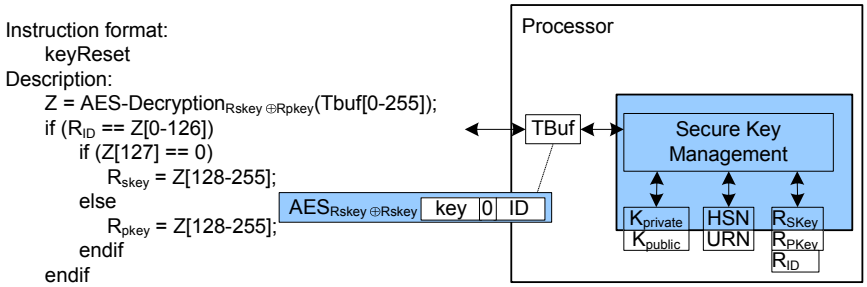

Fig. 9. The logic of instruction `keyReset`.

HSMP update public session key. The HSMP unit decrypts the 128-bit encrypted block in `TBuf`. If the ID in the decrypted block *mismatches* the onchip ID, then the first 128 bits of the input are used as a seed to update the current session key. Similar as above, the 128-th bit guide whether R_{Skey} or R_{Pkey} is to be overwritten.

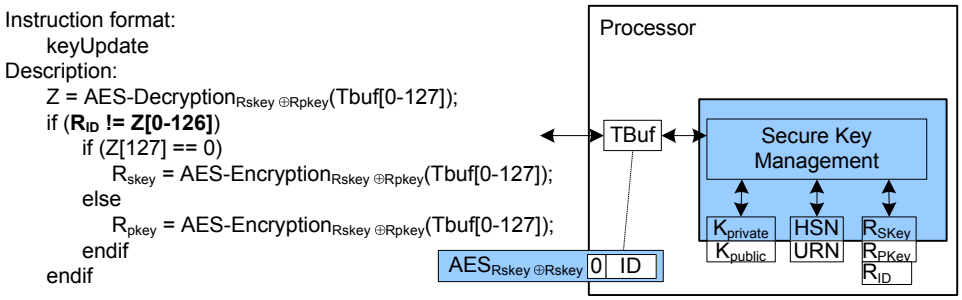

Fig. 10. The logic of instruction `keyUpdate`.

To enforce secure HSMP on the proposed secure processor, the compiler needs to be adapted to use above instructions in code generation. Since these instructions take multiple cycles to complete, and they operate on onchip cryptographic unit, it is possible to overlap their execution with other instructions for performance improvement.

4 Implementing Different HSMP Policies

In this section, we discuss the implementation of different HSMP policies respectively. We use the example in Figure 1. The server broadcasts the video to its users over the Internet. Under the fine-granular billing plan, a user may join and leave the user group freely at which point the server needs to update its keys.

4.1 HSMP-1

To implement HSMP-1 on privacy-aware secure processors, we need two new instructions `tknGen` and `keyInit`. Next we describe the actions that the server and the users need to take to update the session key.

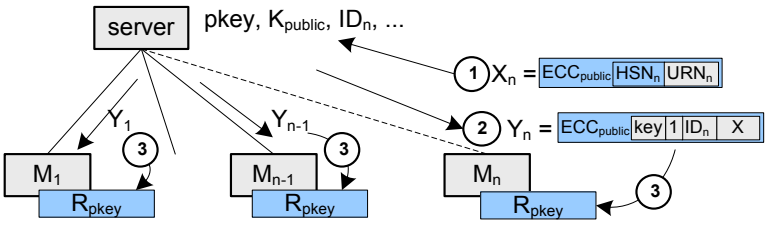

Fig. 11. Implementing HSMP-1.

Member join. When a new user M_n decides to join e.g. pay to watch a video online,

`STEP 1.` M_n first uses the `tknGen` instruction to generate a secure block X_n from his secure processor. The block X is used as his anonymous token to receive the service. A join request is then sent to the server using the point-to-point secure communication. The encrypted block X_n together with the processor batch information are also sent along the request.

`STEP 2.` After receiving the join request, the server needs to consult a trusted database to determine the public key for the corresponding processor batch. The server then selects a not used identifier ID_n and assigns it to the M_n, and picks up a random 128-bit value as the new session key.

For each legal user in the group, the server returns the encrypted data block Y_i ($1 \leq i \leq n$) that contains the received X_i, the assigned identifier ID_i, and the session key. The control bit in the message is set to "1" indicating the key is to be used as a public session key.

`STEP 3.` Upon receiving the block Y_i, each user M_i uses the instruction `keyInit` to extract the ID and session key. The public session key can then be used to decrypt received video packets.

Member leave. Next let us discuss the steps when a user M_n wants to leave i.e. *stop being billed*.

`STEP 1.` M_n sends his ID_n in a *leave* request to the server. It is sent through a secure point-to-point communication.

`STEP 2.` When the server receives such an request, it selects a new session key and prepares a Y_i for all nodes but M_n, that is $1 \leq i \leq (n-1)$. Then Y_i is sent to each user independently.

`STEP 3.` Upon receiving the encrypted block, each remaining user extracts the new session key and continues the video decryption.

Privacy and security analysis We next show that HSMP-1 not only is secure but also protects user privacy.

To hide his identity, a user can use different URNs and generate different tokens (Xs) when he purchases online. The server cannot identify if these Xs are from the same processor as a secure processor never releases the HSN from received Xs.

The hardware access control cannot be bypassed in HSMP-1. To load the session key, the secure processor needs to validate if the HSN in X matches the processor HSN. It is not beneficial to the attacker if he picked up a random X rather than a real X generated from a secure processor. This is because, the data block Y returned from the server is encrypted using the public key of a

real processor batch. If a fake X is used, the returned ID and key information cannot be loaded to his processor, either because Y cannot be properly decrypted (when the attacker tries to upload to another processor batch), or the HSN does match the message HSN in X. It is also not beneficial if the returned Y is released to an illegal user as his processor cannot pass the hardware assisted access control and thus cannot extract the session key.

Advantages and Disadvantages HSMP-1 only requires two instructions to enforce the tamper resistant integration of ECC cryptography implementation, which has already been included in existing secure processors designs [4,14,17]. Therefore the introduced hardware overhead is minimal. Another advantage of HSMP-1 is that, with tamper resistant invisible session keys, legal users can easily split the video processing task from the key management. For example, with a dual-core chip-multiprocessor, the user and the OS has the flexibility to let one core performs key management while the other plays videos.

HSMP1 has some drawbacks. Since the scheme always utilizes the expensive ECC-based cryptography, the server needs to generate a different packet for each of its current users and thus may create a bottleneck in the system.

4.2 HSMP-2

Using the same example, we next describe how to implement HSMP-2 in which the server only needs to send O(1) key update packets to update all users. HSMP-2 divides the key management into three phases — the subscription, the join and the leaving phases.

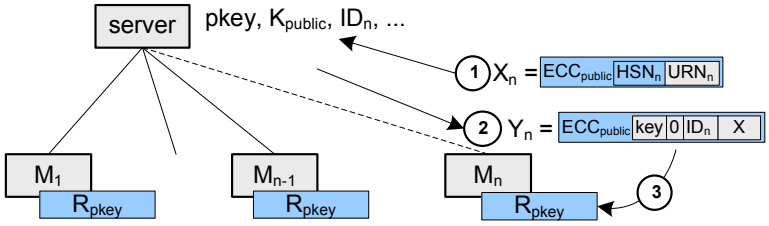

Fig. 12. The subscription phase of HSMP-2.

4.2.1 Subscription

`STEP 1.` The new user M_n prepares an anonymous token X_n using the `tknGen` instruction, and sends it to the server in a join request together with the processor batch information.

`STEP 2.` After receiving the subscription request, the server assigns a not used identifier ID_n to the client. The returned encrypted block Y_n contains

the received X_n, the assigned identifier ID_n, the **secret** session key to be used. The control bit is cleared indicating a private session key to be used.

`STEP 3.` Upon receiving Y_n, the user uses instruction `keyInit` to extract his ID and private session key.

Stable state. The secret session key returned in Y_n is the same one that the server sent to other users. Therefore after the subscription, the group reaches a stable state as follows. Each member has a unique ID while all members share a common *secret* session key. All members except M_n form the current active group who share the same *public* session key as well. M_n has the *secret* session key but not the *public* one. Non-member users do not have any key. While the ID and public session key are visible to the end user, the *secret* session key is invisible and is used for updating the public session key only.

Security analysis. The subscription phase is similar to the join case in HSMP-1 except only one Y_n is generated and the secret key is used. Similar analysis shows that this phase is secure and protects user privacy.

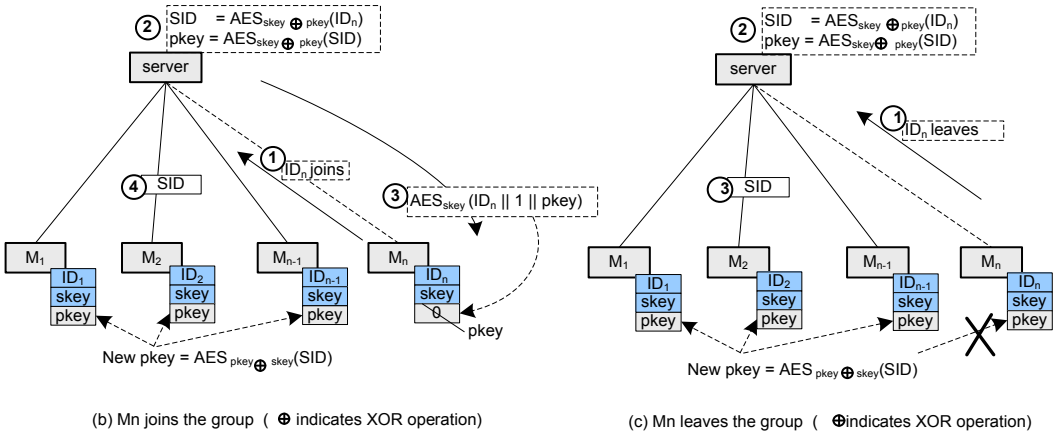

Fig. 13. When a member joins or leaves in HSMP-2.

4.2.2 Member join

A member can only joins the group and receive the service after the subscription phase. At this time, he has a unique ID and the secret session key. The following steps are taken to synchronize all members.

`STEP 1.` A subscribed member sends a join request which may be omitted if the user has already sent out the subscription request. The member includes his ID in the request.

`STEP 2.` When the server receives the request, it generates the new public session key as follows.

$$\text{SID}=\text{AES\_Encryption}_{(skey \oplus pkey)}(\text{ID}_n)$$

$$\text{pkey} = \text{AES\_Encryption}_{(skey \oplus pkey)}(\text{SID})$$

`STEP 3.` The server sends the new session key back to the new member in an encrypted data block $\text{AES\_Encryption}_{skey \oplus 0}(\text{ID}_n||1||\text{pkey})$.

After receiving this block, the new member M_n uses `keyReset` instruction to reset his public session key. Note that the public session key on M_n is zero before the operation.

`STEP 4.` The server also broadcasts SID to all other members i.e. the members before ID_n's join. Each member, upon receiving SID, uses `keyUpdate` instruction which updates the encryption key locally.

$$\text{R}_{pkey} = \text{AES\_Encryption}_{(skey \oplus pkey)}(\text{SID})$$

At this point, the encryption key is the same across all members and the server, which ensures the correct video decryption within the group.

Security analysis. *At STEP 1 of the join case*, it is secure as the communication is through secure point-to-point communication.

At STEP 3 of the join case, the inclusive access control performed by the *keyReset* instruction ensures that only when ID_n is installed, the new session key can be set. Even if M_n may subscribe several times and collects several different IDs, it cannot get the new key if another ID is used. Since the secret key and ID are uploaded in a bundle, and the public key is needed to get the group key, an adversary cannot get the new key even he has a different or the same ID_n from another application.

At STEP 4 of the join case, we see that SID is the ciphertext encrypted with a key created from the current group key, and group key will be updated whenever there is a join and leave. Even the same member e.g. M_n will generate different SIDs if he joins and leaves several times. Other group members may not collect private information about M_n.

To update the public session key from the key update message that contains SID, we need both secret and public session keys. Other than the server, only active members have both keys. An inactive member, even he is subscribed, may not get the new session key due to missing the current public session key. Unsubscripted members do not have the secret key and thus cannot get the new session key either.

4.2.3 Member leave

When a member (with ID_n) decides to leave the active group, HSMP-2 follows the following steps.

`STEP 1.` The leaving member sends the request with his ID to the server.

`STEP 2.` When the server receives the request, it updates the encryption key similar to the join case.

$$\mathrm{SID}=\mathrm{AES\_Encryption}_{(skey \oplus pkey)}(\mathrm{ID}_n)$$
$$\mathrm{pkey} = \mathrm{AES\_Encryption}_{(skey \oplus pkey)}(\mathrm{SID})$$

The server then broadcasts the SID to all members indicating the encryption key should be changed.

`STEP 3.` When remaining members receive SID, they use `keyUpdate` instruction to get their current public session key updated. The leaving member can also receive SID, but he cannot update his key due to the matched ID.

Security analysis. *At STEP 1 of the leave case*, it is the same as the join case. The channel is ensured secure from point-to-point communication.

At STEP 3 of the leave case, similar to step 4 in the join case, inactive and non-subscribed members cannot get the new session key. While the leaving processor may get the message, the exclusive access control mode of `keyUpdate` instruction prevents M_n from updating his session key. M_n may not ignore SID or manipulate a faked SID as otherwise the generated key will be different from others.

The collusion of M_n, inactive and non group members does not work as inactive and non group members have less information than M_n. If M_n cannot get the key, neither do those members.

4.3 HSMP-3 and HSMP-4

The implementation of HSMP-3 and HSMP-4 are similar to HSMP-1 and HSMP-2 respectively. The only difference is that the control bit in the message from the server is set to "0", then the user processor will load the key information to secret session key register instead of the public one.

5 Performance analysis

5.1 Hardware cost

The high level security protection provided by hardware assisted SMP (HSMP) is at the cost of hardware investment and ISA extensions. Next we discuss the cost to integrate different crypto-units onchip. With the fast technology advance, a single chip can hold billions of transistors, prompting the trend to designing Security Processing instruction set eXtension (SPX) due to pervasive security requests from various types of applications. As an analogy example, MMX ISA extension [24] is evolved from widespread multimedia application requirements.

A secure processor usually needs to support both asymmetric cryptographic algorithms e.g. ECC, RSA and symmetric algorithms such as AES, DES. While asymmetric algorithms are more expensive, their key distribution is simpler compared to that in symmetric ones. Symmetric algorithms are used more often such that integrating a hardware implementation can greatly boost up the performance. On the other hand, it is more cost-effective to embed the ECC code onchip and enforce operation protection to defend possible leakage of intermediate results. Recent implementation of ECC algorithms on Pentium IV 2.1GHz processors showed that the delays of ECC encryption and decryption operations are around 5.6ms and 4.1ms respectively [29]. AES hardware implementation has been constantly improved by recent research and industrial effort: ASICS.ws introduced an AES IP core that needs 22-26 cycles at 266Mhz [30]; Schaumont et al. [31] introduced an AES prototype that requires 14 cycles at 154 Mhz.

The space overhead comes from two aspects. One is the onchip secure registers and buffers. Except the key pair and the temporary buffer, all others are 128-bit wide. The total size is around 5K bit, i.e. 640 bytes. The other space overhead is the ECC code embedded onchip and the AES implementation. As we discussed, we would like to have our implemented as an extension of secure processor models rather than a stand-alone design. Much of such overhead therefore can be shared by multiple secure architectural enhancements, including the PKI key pair, the AES implementation, and the ECC code space. In summary we consider that our privacy aware secure processor design has very moderate overhead.

5.2 Processing cost

Here we compare the processing overhead of different HSMP policies in terms of the size of messages, the storage overhead, and the cryptographic computation overhead for each update request. Here we omit the communication and authentication overhead at the network level.

Figure 14 shows the size of key update messages. We compare the size instead of the number of messages since multiple messages may be combined. In HSMP-1 and HSMP-3, a new user needs to send a 256-bit message X and receive a 512-bit message Y; each existing user needs to receive a 512-bit message Y. The total size is (N $\times$ $|Y|$ + $|X|$) for the join case and ((N-1) $\times$ $|Y|$ + $|X|$) for the leave case. Here $|X|$ denotes the size of message X. HSMP-2 and HSMP-4 reduce them to 4T and 2T respectively since the same 128-bit SID is sent to all users.

	Message			Storage	
	Join	Leave	Subscription	server	user
(N is the number of current members, T=128 bit.)					
HSMP-1 / HSMP-3	(4N+2)T	(4N-2)T	–	3NT+1	4T
HSMP-2 / HSMP-4	4T	2T	6T	3NT+1	4T

Fig. 14. Comparison of Message Size and Storage.

Figure 14 also shows the storage overhead. Since the server needs to remember the 256-bit X and the 127-bit ID for each user, plus a current session key, the total overhead at the server is (N$\times$ ($|X|$ + $|ID|$) +1). If a user wants to offload the ID, secret session key and the public session key from onchip to the user memory, it takes two 256-bit blocks i.e. 4T space.

Algorithm	Join Case		Leave Case		Subscription	
	server	user	server	user	server	user
N: the number of members; E, D: the encryption and decryption overhead respectively;						
HSMP-1 / HSMP-3	N E_{ECC}	D_{ECC}	(N-1) E_{ECC}	D_{ECC}	–	–
HSMP-2 / HSMP-4	4 E_{AES}	2 D_{AES} or D_{AES}+E_{AES}	2 E_{AES}	D_{AES}+E_{AES}	2E_{ECC}	E_{ECC}

Fig. 15. Comparison of Cryptographic Computation.

Figure 15 compares the cryptographic computation overhead at the server and the user sides. HSMP-1 and HSMP-3 need to encrypt using ECC for each of its users while HSMP-2 and HSMP-4 generate one SID using AES.

In summary, the overhead introduced from HSMP-1 and HSMP-3 introduced is O(N) while from HSMP-2 and HSMP-4 is O(1). HSMP-1 and HSMP-3 use ECC only while HSMP-2 and HSMP-4 use both ECC and AES algorithms.

6 Related work

In this section we discuss related work in addition to that in the introduction section.

Tamper resistant hardware. Tamper resistant hardware has been integrated in many embedded devices, e.g. Smartcards, for providing low cost and effective security protection [6]-[9]. To protect against various attacks especially physical attacks, mechanisms have been integrated for anti-tampering. For example, special circuit can be integrated such that it erases the key information if the processor is unpacked [9]. On the other hand, new attacks are being developed for attacking tamper-resistant hardware. Some have been proven effective for low cost embedded processors and Smartcards, e.g. power analysis [8]. Fortunately, these methods are still not realistic to attack modern high performance processors which integrate billions of transistors.

Secure processors. The secure processor designs [4,14,22] can provide high level protection again various attacks including those from the hardware and from the OS. In XOM model [4], each processor contains a public/private key pair and specially designed architectural components integrated at the chip boundary. A program running on such processors is encrypted with a session key that is protected by the key pair of the processor. Program and data are encrypted outside the processor chip and decrypted after bringing into the processor chip. Later research provides further enhancements in memory protection [23], performance [17,15]. Similar efforts have been advocated in the industry by the Trusted Computing Group (TCG) [27]. Additional effort can be made with the assistance of the OS e.g. NGSCB by Microsoft [22] creates a newly designed, separately managed secure system agent `nexus` from the current execution environment (processes). While it targets at more complex secure control and protection, the design requires significant modifications in both the OS and the lower level architecture. Our design only adds four instructions and therefore is easier to be integrated.

Group key management. Different group key management algorithms have been proposed in the past. They are divided into three categories [13]: centralized, decentralized and distributed. Our approach falls in the first category in which there is a key distribution center for access control and generating/distributing the keys. LKH [19] is a widely used scheme in this category which forms a logical key hierarchy e.g. a logic tree structure. The cost is in

the order of O(logN) when the tree is balanced.

Broadcast Encryption. Other related work includes broadcast encryption [10–12] which focuses on delivering secure information (keys) to group members through broadcasting channels. The design goal is that the coalition of up to `k` outsiders cannot reveal the key. Broadcast encryption targets at the environment with limited processing power, hardware resources, and tamper resistant ability e.g. pay-TV. Key management for broadcast encryption usually does not support real time group update [11].

7 Conclusions

In this paper we proposed the privacy-aware secure processor designs to defend malicious key sharing for streaming media protection over the Internet. We compare the advantages of using different protection policies and elaborate the architectural designs. We present the implementation details of different policies and analyze their security and performance.

References

[1] BSA Global Piracy Study, May 2006. http://www.bsa.org/usa/research/.

[2] DTCP, 5C Digital Transmission Content Protection. http://www.dtcp.com/.

[3] Intel. http://support.intel.com/support/processors/pentiumiii/sb/CS-007579.htm.

[4] D. Lie, C. Thekkath, P. Lincoln, M. Mitchell, D. Boneh, J. Mitchell, M. Horowitz, "Architectural Support for Copy and Tamper Resistant Software," *ACM Ninth International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS), Nov. 2000.

[5] http://www.modchip.com/.

[6] R. Anderson, and M. Kuhn, "Low Cost Attacks on Tamper Resistant Devices," in M. Lomas et al. (ed.): Security Protocols, *Proceedings of the 5th International Workshop*, LNCS 1361, Springer-Verlag, pages 125-136, France, April, 1997.

[7] O. Kommerling, and M.G. Kuhn, "Design Principles for Tamper-Resistant Smartcard Processors," in *Proceedings of the USENIX Workshop on Smartcard Technol ogy (Smartcard '99)*, Chicago, IL, May 1999.

[8] T.S. Messerges, E.A. Dabbish, and R.H. Sloan, "Investigations of Power Analysis Attacks on Smartcards," in *Proceedings of the USENIX Workshop on Smartcard Technol ogy (Smartcard '99)*, Chicago, IL, May 1999.

[9] R. Anderson, M. Kuhn, "Tamper Resistance - a Cautionary Note," In *Proceedings of the Second USENIX Workshop on Electronic Commerce*, November 1996.

[10] A. Fiat, and M. Naor, "Broadcast Encryption," *13th Annual International Cryptology Conference*, 1993.

[11] A. Wool, "Key Management for Encrypted Broadcast," *ACM Transactions on Information and System Security*, Vol.3(2), 2000.

[12] M. Abdalla, Y. Shavitt, and A. Wool, "Key Management for Restricted Multicast Using Broadcast Encryption," *ACM Transactions on Networking*, Vol. 8, No. 4, August 2000.

[13] S. Rafaeli, and D. Hutchison, "A Survey of Key Management for Secure Group Communication," *ACM Computing Surveys*, Vol.35, No.3, pages 309-329, Sept. 2003.

[14] E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "AEGIS: Architectures for Tamper-Evident and Tamper-Resistant Processing," *ACM 17th International Conference on Supercomputing* (ICS), June 2003.

[15] E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "Efficient Memory Integrity Verification and Encryption for Secure Processors," *IEEE/ACM 36th International Symposium on Microarchitecture* (MICRO), Dec. 2003.

[16] Action Alert Newsletter. http://www.thestreet.com.

[17] J. Yang, Y. Zhang, and L. Gao, "Fast Secure Processor for Inhibiting Software Piracy and Tampering," *IEEE/ACM 36th International Symposium on Microarchitecture*, pages 351-360, San Diego, December 2003.

[18] Y. Zhang, J. Yang, Y. Lin, and L. Gao, "Architectural Support for Protecting User Privacy on Trusted Processors," *The Workshop on Architectural Support for Security and Anti-Virus*, in conjunction with the 11th ASPLOS, Boston, 2004.

[19] C.K. Wong, M. Gouda, and S.S. Lam, "Secure Group Communications Using Key Graphs," *IEEE/ACM Transactions on Networking (TON)*, Vol.8, No.1, pages 16-30, Feb. 2000.

[20] J. Burke, J. McDonald, and T. Austin, "Architectural Support for Fast Symmetric-Key Cryptography," in *ACM Ninth International Conference on Architectural Support for Programming Languages and Operating Systems* (ASPLOS), October 2000.

[21] C. Collberg and C. Thomborson, "Watermarking, Tamper-Proofing, and Obfuscation Tools for Software Protection," *IEEE Transactions on Software Engineering*, Vol. 28:8, Aug. 2002.

[22] P. England, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman, "A Trusted Open Platform," *IEEE Computer*, pages 55-62, July 2003.

[23] B. Gassend, E. Suh, D. Clarke, M. van Dijk, and S. Devadas, "Caches and Merkle Trees for Efficient Memory Authentication," *Ninth International Symposium on High Performance Computer Architecture* (HPCA), Feb. 2003.

[24] Alex Peleg and Uri Weiser. " MMX technology extension to the Intel architecture," *IEEE Micro*, 16(4):51–59, 1996.

[25] D. Lie, C. Thekkath, and M. Horowitz, "Implementing an untrusted operating system on trusted hardware", *Proc. of the 19th ACM Symposium on Operating Systems Principles*, pages 178-192, 2003.

[26] B. Schneier, "Applied Cryptography," 2nd Edition, John Wiley and Sons, Inc., 1996.

[27] https://www.trustedcomputinggroup.org/home, Trusted Computing Group.

[28] J. Tygar and B. Yee, "Dyad: A system for Using Physically Secure Coprocessors," *TR CMU-CS-91-140R,* CMU, 1991.

[29] Crypto++ 5.1. http://sourceforge.net/projects/cryptopp/ and http://www.eskimo.com/∼weidai/cryptlib.html.

[30] http://www.asics.ws/doc/aes brief.pdf, ASICS.ws Technical Report.

[31] P.R.Schaumount, H.Kuo, and I.M. Verbauwhede, "Unlocking the design secrets of a 2.29 gb/s rijndel processor," *DAC*, 2002.