Architectural Support for Protecting User Privacy on Trusted Processors

Youtao Zhang ⋆ Jun Yang ‡ Yongjing Lin ⋆ Lan Gao ‡
⋆ Department of Computer Science ‡ Department of Computer Science
The University of Texas at Dallas The University of California at Riverside
Richardson, TX 75083 Riverside, CA 92521

Abstract

Recently proposed trusted processor model is a promising model for building secure applications. While effective designs have been proposed for protecting data confidentiality and data integrity in such environments, an important security criterion – user privacy is usually neglected in current designs. Due to the increasing concern of privacy protection in the Internet era, such deficiency can hinder the adoption of the new model.

In this paper, we identify the threat model to user privacy and propose a new scheme for user privacy protection. In addition to providing the same ability in protecting data confidentiality and data integrity, the new scheme effectively protects user privacy and only introduces very low overhead.

1 Introduction

The explosion of Internet technology has fertilized the growth of security threats to computer systems drastically. However, building an ultra safe computer system is challenging since security attacks may be launched to any system component (e.g. OS, user applications, memory contents, and even system buses), at any time during execution or data transmission, and can be of any type, known or unknown at the time the system was built. Common security threats include stealing program code and data (violating data confidentiality), altering program code and data (violating data integrity), and collecting program execution behavior (violating user privacy) etc.

Recently a promising trusted processor model was proposed [5, 6, 7, 3, 9] in which only the main processor is trusted in building secure applications[1]. In this model, the processor is equipped with public-key infrastructure (PKI) and enhanced with specially designed on-chip architectural units for performing cryptographic operations. All other components such as OS, co-processors and memory banks are not trustworthy. Trusted processors have many advantages, in particular, the secure execution of user applications is independent of the OS. As a commercial OS usually contains millions of lines of code, its high level security is especially hard to maintain. Excluding OS from the design presumably increases the confidence of providing high level of security to program execution.

However current research on trusted processors mainly focuses on protecting data confidentiality [5, 8, 9] and data integrity [3, 8]. While these designs have achieved much success, an important security criterion – user privacy is usually neglected. User privacy can be violated because each trusted processor has a unique public-private key pair and the public key is released to the software vendor during the software distribution phase, the identity of each processor can be distinguished in the network by its public key. Sensitive private data such as shopping preference, etc., may be revealed without the consciousness of the user. User privacy is becoming a serious concern in a networked environment especially the Internet. In the past, it contributes as the major reason to the failure of the introduction of unique serial

[1] It was also named as trusted hardware [6] or secure processor [9].

number in Pentium III processor [4].

In this paper, we propose a new mechanism for protecting user privacy on trusted processors. Instead of embedding a unique key pair in each processor, the new scheme uses the same key pair for a batch of processors. This greatly relieves the burden on the vendor side to verify the public key. The uniqueness of each processor is ensured by a *secret* hardware sequence number. To further protect user privacy, a user selected random number is incorporated which effectively obfuscates the information transferred in the network. While the new scheme provides protection for data confidentiality, data integrity, and user privacy, it only introduces very low overhead to program distribution and execution.

The rest of the paper is organized as follows. We elaborate the threat model to trusted processors in section 2. Section 3 discusses our proposed solution and its protection of user privacy. Section 4 concludes the paper.

2 Trusted Processor Model and its Threat on User Privacy

We base our design on microprocessors that are equipped with the public-key infrastructure (PKI) since the design is becoming a future trend [5, 2, 7]. Let us term it as "trusted processors". Such a chip is installed with a public-private key pair (k_p, k_s), where k_p, is available to the public. A protected software that runs on a trusted processor is encrypted by the distributor using a symmetric cipher with key k (session key). The encryption not only protects the confidentiality of the software algorithm but also guarantees that it can only *run* on the target processor. To communicate the k to the processor, the distributor uses k_p to encrypt it and ships it along with the software. Secure execution of the protected software begins with computing k using k_s which is carried only once but might take a relatively long time, and decrypting instructions using k which is much faster but is carried on every instruction fetched into the processor. In this way, software encrypted for $processor_1$ can not run on $processor_2$ since they have distinct private keys.

The trusted processor adopts efficient mechanisms in protecting the program data confidentiality and providing memory integrity verification. Ensuring confidentiality means to keep data information hidden from anyone for whom it is not intended. This is achieved through data and instruction encryption. Memory integrity verification is to detect if the memory has been tampered with by an adversary. This is accomplished by creating a *hash (MAC)* value for each memory block. Hashing is especially useful in the three types of attacks considered in trusted processors: *spoofing*, *splicing*, and *replay* [5, 3]. Both memory encryption and integrity checking can be carried efficiently, bringing only a little performance overhead [9, 8].

2.1 Privacy Threat

As the advantages of secure processors become more evident, future web based transactions will be likely to incorporate processor's public key information for either strong or weak security requirements. However, it generates serious privacy problem if each secure processor has a unique public-private key pair.

Let us use an example of purchasing a newly developed computer game on-line to illustrate the privacy problem (Figure 1). An end user U purchased a game on-line from a company C and wants to download it to his own computer. Under the trusted processor model, U has to send C his public key K_{public} since C would like to release his encrypted game *only* to the trusted processor that U will use to run the software. C does not want U to see the clear text of the software game because U might be a market competitor. On the other hand, if U is an ordinary game player, U does not want C to collect his private information by remembering his K_{public}. More elaborately:

(1) Company C should not easily trust the public key passed by U since it could be faked by U in which case the private key is known to U. Once U gets the encrypted code, he can simply decrypt the game and analyze it. For this reason, C has to validate the received public key, that is, it is the public key of an existing trusted processor. A possible solution is to query a central database created by the processor manufacturer (Figure 1). However, it is not desirable since

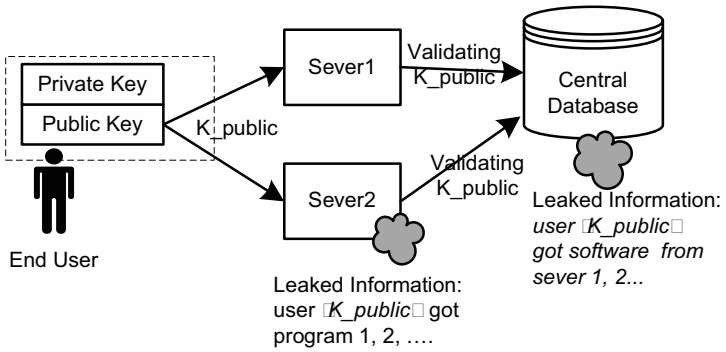

Figure 1: Possible Privacy Leakage.

the processor manufacturer can collect and data mine these queries from which it gets private information it is not supposed to get (even though the company may not release it to the public).

(2) As the public key is unique to each processor, its user has no control over it since otherwise he may not get the session key correctly decrypted. However, C can easily identify a returned user if his public key was used before. After some time, U's activities can be collected and analyzed by C without U's awareness. Further, C can trade U's information with other sales companies to obtain larger pool of customers' information including U's additional activities. This is certainly not what U or any normal customer desires.

Both problems come from the fact that a trusted 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 [4]. Intel tried to have a hardware serial number that is unique and can be queried by some software. Due to widespread privacy concerns, the number is finally *disabled* in the default settings.

3 Proposed Solution

The main functionalities of the public-private key pair are two folds: 1) it provides a mechanism to protect program data confidentiality and integrity; and 2) it ensures uniqueness such that only the designated processor can execute a program. It is easy to see that the second aspect is the reason for potential violation of the user privacy. We therefore propose a technique to conceal the uniqueness of the processor.

Instead of using a unique key pair for each individual processor, we use a batch processor key pair and a hardware pseudo random number (PRN) whose combination uniquely identifies a trusted processor. The PRN will not be seen by the other party but will serve as part of a unique ID within the group of the processors identified by the batch key. This not only fits into our need but also relieves the burden of distributing and reclaiming huge number of unique key pairs. In addition, we allow the user to specify a random number (URN) to further obfuscate the information transferred across the network. This is to discriminate multiple exposures of a single user in an open network.

Figure 2 shows the additional information that should be supported by the hardware. The manufacture embeds a physical PRN into the processor which pertains the same level of secrecy as the key pair. Note that the PRNs are distinct from within the group of processors that share the same key pair. Maintaining such PRNs are much less expensive since the numbers are smaller and they can be reused for different groups. The URNs are defined by the user. They are also inputs to the privacy protection unit (PPU) which executes the communication protocol that we will define later.

Under such a framework, the concern number (1) we addressed earlier is now solved by using the combination of processor batch key pair and the PRN. Introducing the batch key pair for a group of proces-

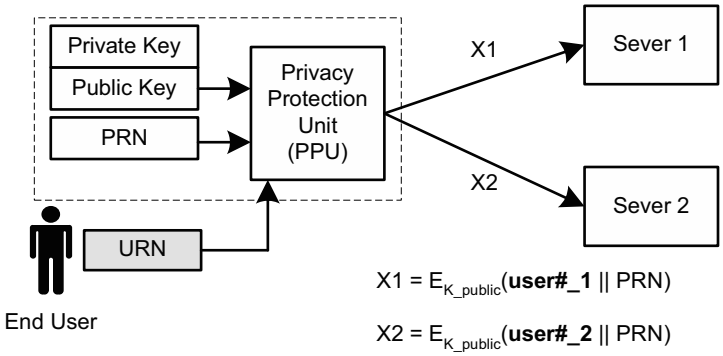

Figure 2: Privacy Protection.

sors significantly reduces the size of the key database maintained by the chip manufacture. The seller company can simply query for a public key by providing, for example, the user processor's model number and serial number. The public key obtained through this way is much trustworthy. On the other hand, using the batch key can prevent the seller from collecting and study user information.

Concern number (2) is solved by incorporating URN into an on-line transaction. Every time the user purchases a product from a vendor, he should specify a random number, preferably unique every time, which is used by the PPU to obfuscate the user's identity. Through this way, the vendor cannot tell whether a series of transactions is from the same person or different people, further protecting the consumer's privacy. An important premise here is that we do not consider the attacks during the network communication. We expect that those attacks be better handled in network protocol layers.

The detailed steps of our solution are shown in Figure 3.

In the first two steps, a user U creates a token consisting of the PRN and a URN so that the value of X varies every time. X is encrypted by U's own K_{public} so that no one else can decrypt it. In step 3, U tells the seller C not only X but also his processor information regarding to the batch identity. This is used in step 4 where C queries an authorized database for the authentic public key. This may or may not be equal to the K_{public} provided by U. In the latter case, the transaction would be broken at step 6 (more later). Once the authentic K_{public} is obtained, C bundles the symmetric key k used to encrypt the software with the token X received from U, together with the digital signature of the entire software and calculates Y. The signature can be used to check of the software has been altered before execution. In step 5, Y is inserted at the head of the software and sent to U. In last step, U needs to peel off two levels of protection to obtain the PRN. Now it is clear that when the public key C obtained from an authority is different from what U provided, step 6 would fail on mismatch of the PRNs.

3.1 Privacy Analysis

Let us now study how user privacy is safely protected. In Figure 3, as long as the URN is different every-time U wants to purchase goods on-line, X will appear differently on the web as if it is a new buyer. Many widely used encryption algorithms such as AES and 3DES are believed to do a good job in generating pseudo-random numbers. Thus, we can rely on PPU to give us a different number every time (or the series repeats itself with a huge cycle time). Thus, an observer can never figure out if two X's are from the same person. As a result, C can not collect the information based on the X it receives every time.

3.2 Security Analysis

The proposed protocol aims to protect both the user and the seller. It protects the user from being traced by the sellers through varying the user identity every time. It protects the sellers in such a way that a fake

Software Distribution Phase:

step 1. U generates a random number URN as shown in Figure 2.

step 2. PPU then calculates $X = E_{public}(PRN||URN)$, i.e., it encrypts the concatenation of PRN and URN using the processor's public key.

step 3. X is then sent to the C, together with processor's batch information.

step 4. C checks the batch information and get the well-known public key K_{public} from processor manufacturer and then generates $Y = E_{public}(X||k||digital\ signature)$

step 5. C attaches Y at the head of the encrypted software.

Software Execution Phase:

step 6. When U receives the software, it sends Y to the PPU where Y is decrypted. After that, X is extracted and decrypted, followed by retrieval of PRN. It then compares PRN with the physical random number stored in the processor. If they match, the execution continues. Otherwise, it stops the execution.

Figure 3: Encryption and Decryption Steps with PRN and URN.

user cannot obtain any sensitive information such as the code of a software or critical data. This is because if a "user" uses a public-private key pair other than that of a secure processor, (s)he will not be able to decrypt Y generated in step 4. Thus, the session key k will not be released and the code can remain securely encrypted.

It should be noted that step 6 in the protocol is performed in an atomic procedure and the results the PPU computed cannot be released to off-chip components just as the processor's private key. This is because all the processors sharing the same public-private key pair can decrypt Y. However, the PPU will guarantee that the session key k and the $digital\ signature$ are destroyed if the PRN (after decrypting the X) does not match its own PRN.

3.3 Hardware Design

As shown in Figure 2, a new architectural enhancement – privacy protection unit (PPU) is proposed to generate X and verify PRN (step 2 and 6 in Figure 3) in trusted processors. These tasks involve RSA based encryption and decryption operations. As these operations are assumed to be much less frequent, RSA may not be implemented in hardware. In this paper, we assume both encryption and decryption operations are done in software using the trusted processor itself. To ensure security, the processor is enhanced to restrict the execution of RSA cryptographic operations: the execution may not be interrupted, the intermediate results are cleared before the end of the operation, and the result is sent to the secure register for further processing.

3.3.1 Space Cost

The scheme introduces very modest space overhead. As PRN is used to distinguish each individual processor, its size depends on how many processors in a batch and 64 bits should be reasonably large to serve the purpose. Other space overhead includes on-chip buffer for storing RSA encryption code (The original model requires the decryption code only), and the control logic. Both will not occupy large space.

3.3.2 Runtime Overhead

More concerns come from the runtime overhead in the new scheme. Secure execution of trusted program can be divided into distribution phase and execution phase. In the former, the user and the server collaborate to get a uniquely encrypted session key. In the latter, the designed trusted processor loads and executes the program.

We found that the runtime overhead should not be too high in both phases. Recent C++ implementation of RSA algorithms on Pentium IV 2.1GHz processors showed that the delays of RSA encryption and decryption operations (with 1024 bit keys) are around 0.1ms and 5ms respectively [1].

In the distribution phase, the end user has to generate a public-key encrypted X from PRN and URN. As encryption delay is less than 1ms, it is generally not a problem since (1) it is a one time delay, (2) the computation can done beforehand, and (3) the network communication delay is usually much high than this.

In the execution phase, the trusted processor has to perform one extra round of decryption to extract PRN in X and compare it to the PRN of the processor, which adds 5 ms before having the plaintext session key ready. It is also not a problem since it happens only when the session key is not present in the processor. Once the session is decrypted, it can be cached on chip even the OS dispatches another program to execute.

4 Conclusion

In this paper, we identify the threat to user privacy on trusted processors and propose new architectural enhancements for defending it. To provide security protection for data confidentiality and data integrity, we use a batch key pair and a secret unique processor random number. While at the same time, the user privacy is protected by integrating a user provided random number in the scheme. The new scheme introduces little overhead to both program distribution and execution.

References

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

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

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] J. Yang, Y. Zhang, and L. Gao, "Fast Secure Processor for Inhibiting Software Piracy and Tampering," *IEEE/ACM 36th International Symposium on Microarchitecture* (MICRO), Dec. 2003.

[10] X. Zhuang, T. Zhang, and S. Pande, "HIDE: an infrastructure for efficiently protecting information leakage on the address bus," *ACM 11thInternational Conference on Architecture Support for Programming Language and Operating Systems*, 2004.