

# Efficient Group Key Management with Tamper-resistant ISA Extensions

Youtao Zhang  
Computer Science Department  
University of Pittsburgh  
Pittsburgh, PA 15260

Jun Yang, and Lan Gao  
Computer Science and Engineering Department  
University of California at Riverside  
Riverside, CA 92521

## Abstract

*We present a tamper-resistant architectural enhancement for secure group key management in group communication applications. Using specially designed four cryptographic instructions, we show that the hardware assisted design can greatly reduce the management overhead to the order of  $O(1)$  in terms of rekey messages, storage cost, and the encryption computation cost.*

## 1 Introduction

The last decade has seen the enormous increase of group oriented applications many of which use secret group keys to secure the sensitive data transmission among scattered group members across the network or even over the Internet. Many group key management algorithms [1] have been proposed to update these keys when the group changes, preventing information leakage prior to and after the change. Simple solutions such as updating new keys through point-to-point channels are inefficient and tend to incur large communication and computation overhead. The widely used logical key hierarchy (LKH) algorithm [2] reduces the overhead to  $O(\log N)$  by saving  $O(\log N)$  auxiliary keys (where  $N$  is the number of group members) on each member to assist the key distribution.

On the other hand, the pervasive security requirement in today's applications has motivated the integration of specially designed cryptographic units on the processor chip. Recent secure processor designs [14, 11, 12, 19] showed that they can effectively defend various attacks including those from the OS and the hardware, alleviating the difficulty and complexity in designing secure systems.

In this paper, we extend the secure architectural design practice to support group oriented network applications. We design an onchip secure key management (SKM) unit and introduce four instructions to access it in the user application. We then present a SKM-assisted group key management algorithm which reduces the management overhead to

the order of  $O(1)$  in terms of rekey messages, storage cost in each member, and the encryption computation cost.

In the rest of the paper, we discuss the architectural design of SKM in Section 2. The group key management algorithm using SKM is presented in Section 3. Section 4 evaluates the performance. The related work is discussed in Section 5. Section 6 concludes the paper.

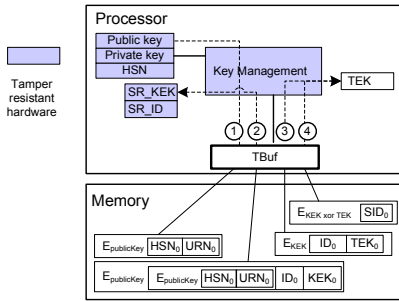
## 2 SKM: Secure Key Management Unit

### 2.1 Security attack model

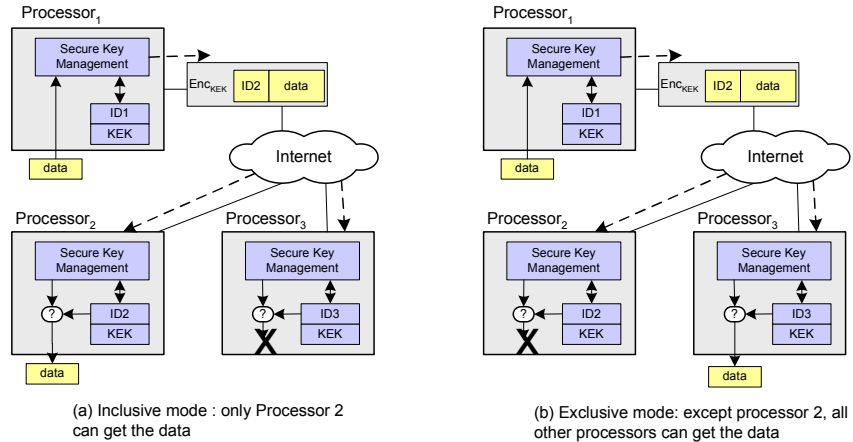
In this section we describe the security attack model. We assume a centralized algorithm setting in which a reliable KDC is responsible for generating new keys. The membership is changing dynamically with current members form a trusted group at runtime. An involved member/computer node/user is trusted when it is within the group and not otherwise. Each member has full control of his own computer. Members may collect information when they are either in or outside the group and try to achieve the advantage of accessing (attacking) the sensitive communication by the time they are not in the group.

For each node in the system, its sensitive information is securely managed by a specially designed key management architectural component. Similar to [14, 17], we assume this component are free from physical attacks.

At the network level, we assume that the network connection is insecure. Packets sent over the network may be dropped or tampered with. We rely on underlying network protocols in detecting these attacks. That is, "rekey" messages can reach legal members, and are signed to ensure integrity. The encrypted messages may also be received by attackers and former group members who can save and manipulate these messages, and communicate them to their local hardware with the goal to break into the group.



**Figure 1. Architectural Design of SKM.**



**Figure 2. Mode of Hardware Assisted Access Control.**

## 2.2 The architecture overview

Figure 1 illustrates the onchip architectural design of SKM which includes multiple tamper-resistant registers, and a secure management function unit. They are accessible through special instructions.

- Two secure registers for storing the private/public key pair of the processor (each key is 1024-bit long). The private key is kept secret. As in [20], the key pair is the same for a batch of processors, e.g. all processors manufactured in September 2004 are of the same keys.
- A unique hardware sequence number (HSN) register. To distinguish an individual processor in the batch, each processor has a unique 128-bit HSN. It is only accessible from the SKM unit i.e. invisible to the public. Together with a user random number (URN) the processor privacy is ensured [20].
- Two secure registers for key management – an ID register (SR\_ID), and a KEK (key encryption key) register (SR\_KEK). While the SR\_ID is visible to the user, the SR\_KEK is kept secret. They are 128-bit registers.

In addition, there is a *semi*-tamper-resistant TEK (traffic encryption key) register. While it is visible to the user, it can only be written by the SKM.

- A 2048-bit buffer for secure management Tbuf. It is to buffer data before writing back to memory or after loading from the memory. It does not need to be tamper resistant.
- A secure key management (SKM) unit. It is integrated for managing secure registers and to support the four newly designed ISA extensions as discussed next.

## 2.3 Tamper resistant ISA extensions

### ISA extension for KEK/ID initialization

- `kekInit R1` (path (1) in Figure 1):

$$X = \text{RSA\_Encrypt}_{\text{public}}(\text{HSN} || \text{R1});$$

$$\text{TBuf}[0-1023] = X.$$

SKM prepares a new secure data block and saves it in TBuf. The block contains the processor HSN and a user-provided random value that is fetched from R1.

- `kekReceive` (path (2) in Figure 1):

$$X = \text{RSA\_Decrypt}_{\text{private}}(\text{TBuf}[0-2047]);$$

$$Y = \text{RSA\_Decrypt}_{\text{private}}(X[0-1023]);$$

$$\text{if } (Y[0-127] == \text{Processor\_HSN})$$

$$\text{SR\_ID} = X[1024-1151];$$

$$\text{SR\_KEK} = X[1152-1279];$$

$$\text{TEK} = 0.$$

SKM loads and decrypts a 2048-bit data block from TBuf. If the check succeeds i.e. the message HSN matches the processor HSN, then the hardware ID and KEK registers are initialized from the decrypted block. The TEK is set to zero.

### ISA extension for group key update

- `tekSet` (path (3) in Figure 1):

$$X = \text{AES\_Decrypt}_{\text{KEK}}(\text{TBuf}[0-255]);$$

$$\text{if } (\text{SR\_ID} == X[0-127])$$

$$\text{TEK} = X[128-255];$$

SKM decrypts a 256-bit encrypted secure data block from TBuf. If the first 128-bit ID matches the onchip ID, then the second half (128 bits) is used as the current TEK.

- `tekUpdate` (path (4) in Figure 1):

```

Input = TBuf[0-127];
X = AES_Decrypt $_{KEK \oplus TEK}$ (Input);
if (X != SR_ID)
    TEK = AES_Encrypt $_{KEK \oplus TEK}$ (Input);

```

SKM decrypts a 128-bit encrypted data block from `TBuf`. If the cleartext matches the current onchip ID, then the execution halts; otherwise, the input is *encrypted one more time* to update the TEK register.

## 2.4 Design issues

**Hardware assisted TEK/KEK management.** There are two types of keys – KEKs and TEKs. In our design, both are secured by the hardware. However the former is invisible while the latter is visible to the user. TEK is preferably in clear text since the end user can thus use the key to decrypt multiple encrypted messages simultaneously on different other machines without sacrificing the security.

**Two access control modes** The secure registers and buffers can be accessed in two modes. The access is granted based on the comparison result of the ID in the message and the ID of the machine.

In Figure 2, assuming a member on processor `Proc1` is to transmit some secure data to others across the network. Using KEK, the data is encrypted in a secure block with 2 fields embedding the target ID and the data. When the encrypted block is sent to the network, all members including the attacker can get it. To ensure security, `Proc1` has the flexibility to specify who can correctly decrypt.

(i) Inclusive delivery mode (Figure 2(a)). Each receiving processor checks to see if its ID matches the one in the encrypted block. The processor `Processor2` can decrypt the data block since its ID matches. Other processors (`Processor3`) do not have matching ID and thus get nothing.

(ii) Exclusive delivery mode (Figure 2(b)). Each receiving processor performs the check as above but the result is used just to the opposite. A matching processor discards the result while other processors proceed to decrypt the data.

## 3 Application: Efficient Group Key Management using SKM

In this section, we present the centralized key management algorithm using SKM.

### 3.1 Overview

Our algorithm contains three phases – subscription, join and leave. Compared to [2], a group member needs to subscribe before receiving the service. In this phase, he securely receives his KEK and assigned ID from the KDC using the point-to-point channel. Once subscribed i.e. with

the KEK/ID assigned, a member may join and leave the group multiple times without new subscription. Of course he may choose to subscribe again for privacy reasons. Another scenario in which subscription may be mandatory is that the group application has been running for a long time. In this case, the KDC has generated a significant large number of KEK/ID pairs and group keys. The KDC may want to initiate a new session and replace the old KEK by pushing current active members to re-subscribe the service.

In the following discussion (Figure 3), we assume a key distribution center (KDC) and multiple group members  $M_1, M_2, \dots, M_n$ .

### 3.2 Case 1: member subscription

The subscription contains three steps (Figure 3(a)).

**STEP 1.** A new member, for example  $M_n$  in the figure, creates its anonymous token  $X$  and sends it to the KDC. This is done using the *kekInit* instruction at  $M_n$  side, and sent through the private point-to-point communication channel between the KDC and  $M_n$ . The encrypted token  $X$  contains both the processor  $HSN_n$  and a random  $URN_n$ . While  $HSN_n$  cannot be changed,  $M_n$  can select a different  $URN_n$  each time he subscribes to the group.

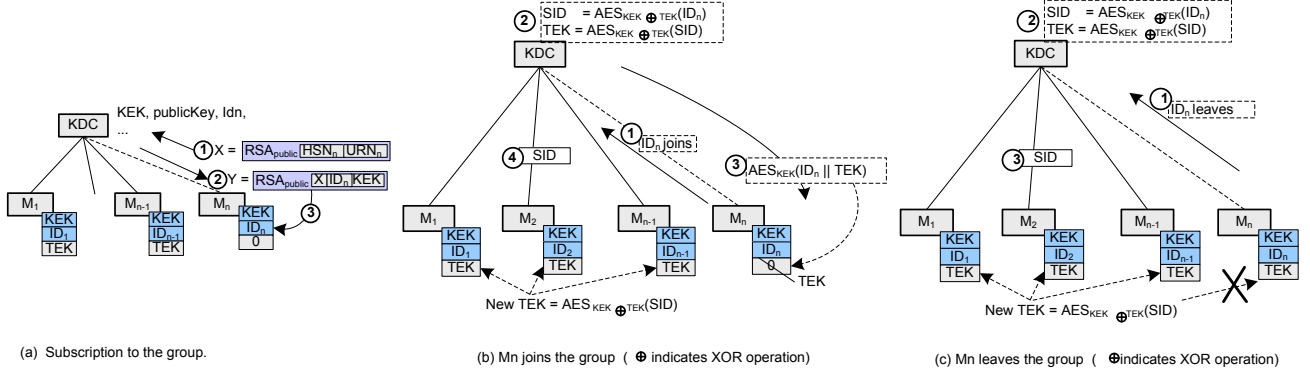
**STEP 2.** After receiving  $X$ , the KDC generates a unique  $ID_n$  that is not used in the group. Using the public key of the processor, the KDC encrypts a block  $Y$  as  $RSA_{public}(X||ID_n||KEK)$  where KEK is currently shared by all group members.  $Y$  is sent back to  $M_n$ .

**STEP 3.** When  $M_n$  receives  $Y$ , it first decrypts  $Y$ , and then decrypts  $X$ . The SKM unit on  $M_n$  compares the HSN contained in  $X$  with the one onchip. If they match, then the ID/KEK are uploaded into the secure registers; and the onchip TEK is reset to 0. This is done using the *kekReceive* instruction.

**Stable Group State** After the subscription, we reach a stable state as follows. Each group member has a unique ID while all members share a common KEK. A subset of subscribed members form the current active group. All active members share a common TEK while non-active members do not have this key. Clearly a newly subscribed member is not in the active group. While ID and TEK are visible to the user, the KEK is kept secret. The group key (TEK) is used for securing group communication.

### 3.3 Case 2: member join

A member joins the group after the subscription i.e. when it has its unique ID and the current secure KEK. In addition to the KEK, the KDC and all active members share the same group key (TEK). If the joining member is the first member in the group, the KDC selects a random TEK as its current TEK. When a new member  $M_n$  is to join the group, the following steps are taken.



**Figure 3. Subscription, Join and Leave of an Individual Member.**

STEP 1. A subscribed member  $M_n$  sends a join request to the KDC. If  $M_n$  already sends out the *subscription* request, this *join* request may be omitted. Otherwise,  $M_n$  sends his ID to the KDC using the private communication channel.

STEP 2. When the KDC receives the request, it generates the new group key as follows

$$\begin{aligned} \text{SID} &= \text{AES}_{\text{KEK} \oplus \text{TEK}}(\text{ID}_n) \\ \text{TEK} &= \text{AES}_{\text{KEK} \oplus \text{TEK}}(\text{SID}) \end{aligned}$$

The first computation generates an anonymous ID for updating the key while the second one generates the new group key.

STEP 3. The KDC then sends the group key back to  $M_n$  in an encrypted data block with inclusive access control mode i.e.  $\text{AES}_{\text{KEK}}(\text{ID}_n || \text{TEK})$ .

When  $M_n$  receives the block, it executes the *tekSet* instruction which compares the ID onchip with that contained in the received block. Upon a match, the key is loaded into the TEK register as the current group key.

STEP 4. The KDC also broadcasts the SID to update the group key of active members.

When an active member receive the SID, he executes the *tekUpdate* which updates as follows

$$\text{TEK} = \text{AES}_{\text{KEK} \oplus \text{TEK}}(\text{SID})$$

The group key TEK is now the same across all active members including the KDC and the new active member.

### 3.4 Phase 3: member leave

When a member  $M_n$  is to leave the group, he has a unique ID, the current secure KEK, and the current group key TEK. The following steps are taken by the group.

STEP 1.  $M_n$  sends a request to KDC indicating his leave from the group. The request includes his unique  $\text{ID}_n$ .

STEP 2. When the KDC receives the request, it updates the current group key similar to that in the join case.

$$\begin{aligned} \text{SID} &= \text{AES}_{\text{KEK} \oplus \text{TEK}}(\text{ID}_n) \\ \text{TEK} &= \text{AES}_{\text{KEK} \oplus \text{TEK}}(\text{SID}) \end{aligned}$$

The KDC broadcasts the SID to all active members including the leaving member.

Upon the receiving of the SID by an active member, he executes the *tekUpdate* instruction which updates the group key as follows

$$\text{TEK} = \text{AES}_{\text{KEK} \oplus \text{TEK}}(\text{SID})$$

After the update, the TEK is the same on all members and the KDC.

## 3.5 Security analysis

Due to space limit, the security analysis of the proposed scheme is omitted. Details can be found in [21].

## 4 Performance Evaluation

In this section we compare the proposed SKM algorithm to two widely used software based algorithms [2] – LKH uses a logical key tree graph in which members are organized in a tree structure; and STAR uses a flat structure in which all members are at the same level. The results show that with secure architectural support, SKM greatly reduces the key management overhead and complexity, justifying the investment in hardware especially given the fact that many cryptographic units (AES unit) either have already been or are to be included in the modern processors [14, 11].

The following lists parameters used in this section.

- N is the number of current group members;
- K is size of the group key (128 bits);
- For LKH, the key tree is balanced and with degree 4;
- D/E denotes the AES decryption/encryption overhead;
- $\text{RSA}_{size}$  denotes the key length of RSA;
- $\text{RSA}_{enc/dec}$  denotes the RSA decryption/encryption overhead.

**Hardware cost.** Hardware AES implementation provides much better runtime performance [22, 23] and thus increasingly integrated into modern processors [18, 14]. Due

to infrequent invocation, RSA code is embedded onchip and executed under secure mode. Thus the extra hardware cost of SKM includes several onchip secure registers and buffers which is estimated to be around 5K bits (i.e. 640 bytes). Costs, such as the PKI key pair, the AES implementation, and the RSA code space, are shared with other secure architectural designs.

Alg.	Join	Leave	Subscription
Star	2K	(N-1)K	–
LKH	$(\log_4 N + 1)K$	$2(\log_4 N)K$	–
SKM	3K	K	$3\text{RSA}_{size}$

Figure 4. Comparison of Message overhead.

**Size of messages.** We first compare the size of rekey messages. We compare the size instead of the number of messages since multiple messages may be combined to form a large single message. The comparison in Figure 4 shows that SKM has message size overhead in the order of  $O(1)$ . It has the smallest message size in the leave case. When the group has more than 16 nodes, the size in the join case is smaller than LKH.

SKM has an additional subscription phase, which is performed only once when a member first subscribes into the group. The message size is 3072 bits – the message sent to the KDC is 1024 bits and the one received from the KDC is 2048 bits. While it is expensive, this cost is a one-time cost and can be amortized from multiple join/leave requests.

Alg.	KDC	Member
Star	NK	K
LKH	$(2N-1)K$	$(\log_4 N)K$
SKM	$(N+1)K$	2K

Figure 5. Comparison of Storage overhead.

**Storage overhead.** We next compare the additional storage for key management. We omit the storage of the current group key (TEK) as it is required in every algorithm. The comparison in Figure 5 shows that in SKM the KDC needs  $(N+1)K$  storage —  $N$  unique IDs and one KEK. Each member only needs 2K storage — the ID and the KEK.

**Cryptographic computation.** We then compare the cryptographic operations performed on the KDC and the member nodes Figure 6. Additional RSA encryption and authentication operations may be needed to ensure the message is received correctly (at the network level). We omit this as it is applicable to all algorithms.

The comparison in Figure 6 shows that in SKM the KDC performs two encryption operations to update the group key. Two additional encryption operations are required in the

join case as a message of 2K size is encrypted and sent to the new member. Each new member performs two decryption operations to get the new group key while other active members perform one decryption operation to verify the ID and one encryption operation to update the TEK. In the subscription phase, the KDC has to perform one RSA encryption for a message of 2048 bits while the joining member performs one RSA encryption before sending and two RSA decryption operations after receiving the response back from the KDC.

## 5 Related Work

*Group key management and broadcast encryption.* Various group key management algorithms have been proposed in the past. They are divided into three categories [1]: centralized, decentralized and distributed. Our approach falls in the first category in which there is a key distribution center (KDC) for access control and generating/distributing the keys. Other related work includes broadcast encryption [8, 9] which focuses on delivering secure information (keys) to group members through broadcasting channels.

*Secure processors and tamper resistant hardware.* Recent research in the computer architecture community proposes secure processor designs [14, 11, 19, 12, 15] for providing high level protection against various attacks including those from the hardware and from the OS. We extend the design to support group oriented application in this paper.

Tamper resistant hardware has been integrated in many embedded devices, e.g. Smartcards, for providing low cost and effective security protection [3, 4]. While new attacks are being developed to attack such devices, protecting mechanisms have been advanced as well e.g. special circuit can be integrated such that it erases the key information if the processor is unpacked [5]. Attacks targeting at modern high performance processors [6, 7] are much more complicated and subject to be simple defenses e.g. software obfuscation [10].

## 6 Conclusions

In this paper we proposed the tamper-resistant key management (SKM) unit with ISA extensions. Using SKM we designed an efficient centralized key management algorithm. Our analysis showed that without considering the subscription cost, the SKM-assisted key management overhead is reduced to  $O(1)$  in terms of rekey messages, storage cost, and the encryption computation cost

## References

[1] S. Rafaei, and D. Hutchison, “A Survey of Key Management for Secure Group Communication,” *ACM Computing Surveys*, Vol. 35(3):309-329, Sept. 2003.

Algorithm	Join Case		Leave Case		Subscription	
	KDC	Member	KDC	Member	KDC	Member
STAR	NE	D	NE	D	-	-
LKH	$3(\log_4 N)E$	$(\log_4 N+1)D$	$2(\log_4 N)E$	$(\log_4 N)D$	-	-
SKM	4E	2D or D+E	2E	D+E	$2 \cdot \text{RSA}_{enc}$	$\text{RSA}_{enc}+2 \cdot \text{RSA}_{dec}$

**Figure 6. Comparison of Cryptographic Computation.**

- [2] C.K. Wong, M. Gouda, and S.S. Lam, "Secure Group Communications Using Key Graphs," *IEEE/ACM Trans. on Networking*, Vol.8(1):16-30, 2000.
- [3] R. Anderson, and M. Kuhn, "Low Cost Attacks on Tamper Resistant Devices," in M. Lomas et al. (ed.): Security Protocols, LNCS 1361, pp. 125-136, France, April, 1997.
- [4] O. Kommerling, and M.G. Kuhn, "Design Principles for Tamper-Resistant Smartcard Processors," in *Smartcard '99*, May 1999.
- [5] R. Anderson, M. Kuhn, "Tamper Resistance - a Cautionary Note," In *the Second USENIX Workshop on Electronic Commerce*, November 1996.
- [6] D. Boneh, and D. Brumley, "Remote Timing Attacks are Practical," *USENIX Security Symposium*, pp. 139-146, 2005.
- [7] A. Shamir, and E. Tromer, "Acoustic cryptanalysis," <http://www.wisdom.weizmann.ac.il/tromer/acoustic/>.
- [8] A. Fiat, and M. Naor, "Broadcast Encryption," *13th Annual International Cryptology Conference*, 1993.
- [9] A. Wool, "Key Management for Encrypted Broadcast," *ACM Trans. on Information and System Security*, Vol.3(2), 2000.
- [10] C. Collberg and C. Thomborson, "Watermarking, Tamper-Proofing, and Obfuscation Tools for Software Protection," *IEEE Trans. on Software Engineering*, Vol. 28(8), Aug. 2002.
- [11] P. England, B. Lampson, J. Manferdelli, M. Peinado, and B. Willman, "A Trusted Open Platform," *IEEE Computer*, pp. 55-62, July 2003.
- [12] B. Gassend, E. Suh, D. Clarke, M. van Dijk, and S. Devadas, "Caches and Merkle Trees for Efficient Memory Authentication," In *HPCA*, Feb. 2003.
- [13] A. Peleg and U. Weiser. "MMX technology extension to the Intel architecture," *IEEE Micro*, 16(4):51-59, 1996.
- [14] D. Lie, C. Thekkath, P. Lincoln, M. Mitchell, D. Boneh, J. Mitchell, M. Horowitz, "Architectural Support for Copy and Tamper Resistant Software," In *ASPLOS*, Nov. 2000.
- [15] R.B. Lee, P. Kwan, P. McGregor, J. Dwoskin, and Z. Wang, "Architecture for Protecting Critical Secrets in Microprocessors," *ACM ISCA*, 2005.
- [16] <https://www.trustedcomputinggroup.org/home>, Trusted Computing Group.
- [17] E. Suh, D. Clarke, B. Gassend, M. van Dijk, and S. Devadas, "AEGIS: Architectures for Tamper-Evident and Tamper-Resistant Processing," In *ICS*, June 2003.
- [18] J. Tygar and B. Yee, "Dyad: A system for Using Physically Secure Coprocessors," *TR CMU-CS-91-140R*, CMU, 1991.
- [19] J. Yang, Y. Zhang, and L. Gao, "Fast Secure Processor for Inhibiting Software Piracy and Tampering," In *MICRO*, Dec. 2003.
- [20] 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.
- [21] Y. Zhang, J. Yang, and L. Gao, "Efficient Group Key Management with Tamper-resistant ISA Extensions," Technical Report, University of Pittsburgh.
- [22] Crypto++ 5.1. <http://sourceforge.net/projects/cryptopp/> and <http://www.eskimo.com/~weidai/cryptlib.html>.
- [23] P.R.Schaumont, H.Kuo, and I.M. Verbauwhede, "Unlocking the design secrets of a 2.29 gb/s rijndel processor," *DAC*, 2002.