UTrustDisk: An Efficient Data Protection Scheme for Building Trusted USB Flash Disk

Yong Cheng, Jun Ma, Jiangchun Ren, Songzhu Mei, and Zhiying Wang
School of Computer Science and Technology
National University of Defense Technology
Changsha 410073 - China
[e-mail: ycheng@nudt.edu.cn]
*Corresponding author: Yong Cheng

Received August 13, 2015; revised March 19, 2016; accepted July 19, 2016; published April 30, 2017

Abstract

Data protection of removable storage devices is an important issue in information security. Unfortunately, most existing data protection mechanisms are aimed at protecting computer platform which is not suitable for ultra-low-power devices. To protect the flash disk appropriately and efficiently, we propose a trust based USB flash disk, named UTrustDisk. The data protection technologies in UTrustDisk include data authentication protocol, data confidentiality protection and data leakage prevention. Usually, the data integrity protection scheme is the bottleneck in the whole system and we accelerate it by WH universal hash function and speculative caching. The speculative caching will cache the potential hot chunks for reducing the memory bandwidth pollution. We adopt the symmetric encryption algorithm to protect data confidentiality. Before mounting the UTrustDisk, we will run a trusted virtual domain based lightweight virtual machine for preventing information leakage. Besides, we prove formally that UTrustDisk can prevent sensitive data from leaking out. Experimental results show that our scheme's average writing throughput is 44.8% higher than that of NH scheme, and 316% higher than that of SHA-1 scheme. And the success rate of speculative caching mechanism is up to 94.5% since the access pattern is usually sequential.

Keywords: Integrity Verification, Data Leakage Prevention, Removable Storage, Hash Trees, Speculative Caching

A preliminary version (DOI: 10.1109/TrustCom.2011.49) of this paper appeared in TrustCom 2011, November 16-18, 2011, Changsha, China. This version includes a detailed description of the data leakage prevention and a security analysis. Besides, an enriched experimental measurement is appended to illustrate the advantage of the efficient data protection scheme. This research was supported by the National Natural Science Foundation of China under Grants No. 61303191 and No. 61402508. We express our thanks to Wangyi Han who checked our manuscript. The authors also gratefully acknowledge the anonymous reviews' advices to the work, plus the editor's work for shepherding the paper through the reviewing process.

1. Introduction

Nowadays, the USB flash disks are widely used for their portability, large capacity and cheap price. Although these disks are playing an important role in data transferring and storing, they also raise new threats in information security. Since the flash disk may carry private and confidential data, even ordinary attacks may cause serious data tampering and information leakages [1]. Take the data integrity verification for example, it is known that data integrity protection is necessary and has been an utmost important issue in building trusted storage system [2]. However, the possibility of the flash chip being accessed directly by general equipment can lead to an easy compromising of the external storage by both software and hardware tampers [3].

In this study, we will discuss how to achieve a trust-based USB flash disk, named UTrustDisk [4][5]. UTrustDisk is a trust-based intelligent USB flash disk which offers data integrity, confidentiality and privacy protection. It is made up of a controller and a flash chip. The controller is actually an USBKey chip that has the similar function to the CPU in computers. The flash chip is a normal chip without special protecting mechanisms and we name it memory, as a more general designation, in the rest of the paper. More details about UTrustDisk are presented in the next section.

We need a trusted base in order to build a trusted storage device. Taking the UTrustDisk as the study case, we assume that the USBKey chip is trusted and the limited on-chip flash, SRAM and EEPROM cannot be tampered. Besides, all the operations (such as hash and encryption) are well protected and thus are safe from attacks. This is a reasonable assumption since a lot of protection technologies are applied to USBKey industry [6]. The external flash is untrusted and can be corrupted by special tools or viruses. In our scheme, the data stored in the untrusted flash is encrypted first and then verified by hash trees. The symmetric key and the roots of hash trees are kept in protected storage.

Hash trees [7] and Message Authentication Codes (MACs) [8] have been widely used in data integrity protection [9][10][11]. But the efficiency of these methods has become the main obstacle for applying in industry practice. More importantly, we need to deploy secure mechanisms on ultra-low-power devices such as flash disk devices [12]. Therefore, we need to improve the efficiency of the data integrity verification and this can be gained via the energy scalable universal hash function [13] and speculative caching. An efficient data authentication protocol will be proposed in Section 3 later.

Data confidentiality is protected by symmetric encryption in UTrustDisk. As the symmetric data encryption algorithm is developed and mature, the remaining work is how to implement the encryption. We have implemented the data encryption on two different levels of security: strict encryption and fast encryption. The strict encryption means that all the data encryption and decryption operations are done inside the UTrustDisk device, in other words, they are operated by the USBKey chip. Given the limited processing capacity of the USBKey chip, we also propose a fast encryption mode, which carries out the encrypting task on the host computer. The symmetric encryption is performed transparently. In other words, the user of UTrustDisk cannot fell the existence of data encryption. We use only one key for encrypting and decrypting, and the key is storage insider the USBKey chip's protected storage.

UTrustDisk has achieved data leakage prevention by adopting Trusted Virtual Domains (TVDs) [14] and virtualization based separation [15]. The TVD is an isolated, transparent and

credible operating domain which can process sensitive data securely. In respect that the TVD is implemented by virtual machines and the UTrustDisk's Resources are limited, we use Feather-weight Virtual Machine (FVM) [16] in our scheme. Although the TVD mechanism can prevent the sensitive data from leakage, there is still a risk that some malicious host may break it down. Therefore, UTrustDisk will monitor the state of the TVD mechanism. Before mounting the storage, UTrustDisk will check the running state of the TVD mechanism and decrypt the ciphertext if and only if the TVD mechanism is running correctly. After mounting the storage, UTrustDisk will monitor the running state of the TVD mechanism. Once UTrustDisk detect that the TVD mechanism is interfered, it will terminate all accesses. The details of data leakage prevention will be discussed in Section 4.

The main contribution is that we proposed a low energy, high efficiency data authentication protocol and a data leakage prevention scheme. Compared with other existing USB flash disk protection technologies such as Armordisk [17], UTrustDisk has protected almost all aspects of data security. And the comparative experiment results show that our scheme is highly efficient and more practical.

The rest of the paper is organized as follows. In the next section we describe previous works and some related researches as the background. In section 3, we go on to discuss authentication algorithm for efficient memory integrity protection. Section 4 presents data leakage prevention mechanisms and security analysis. The performance evaluation of data integrity protection is presented in section 5. And we concluded in Section 6.

2. Background

In this section we introduce the previous work from three aspects: UTrustDisk, hash trees and universal hashing.

2.1 UTrustDisk

In order to address flash disk's data security flaws, we have designed the UTrustDisk. **Fig. 1** gives the architecture of UTrustDisk illustrating that it consists of hardware and software. This device's hardware is a combination of control chip, USBKey chip, and flash chip. And the software is composed of Chip Operating System (COS) and Control Software (CS). Both COS and CS are stored in chip memory and can't be modified without special tools. The vital procedures, e.g. data integrity verification and strict-mode encryption, are implemented in COS. While the CS handles other critical operations such as TVDs, user authentication and fast-mode encryption.

The on-chip memory includes three parts: SRAM, FLASH and E2PROM. These memories can be treated as trusted storage according to our assumption before. The SRAM unit is used for storing temporary data of (from) CPU. The FLASH unit is usually used to store programs, libraries and data. The data stores in FLASH should not be changed frequently because its write operation speed is slow. The EEPROM unit is used for storing programs or other data which may be changed more frequently than FLASH.

UTrustDisk can provide data integrity, confidentiality and data leakage prevention. After the control software starts up, an Isolated Execution Environment (IEE) is built for the sake of executing untrusted processes. The data privacy is protected by IEE via file system filter, network filter and memory filter. The data confidentiality is protected by encryption. The data integrity protection is also achieved in control chip, and we will discuss the details later.

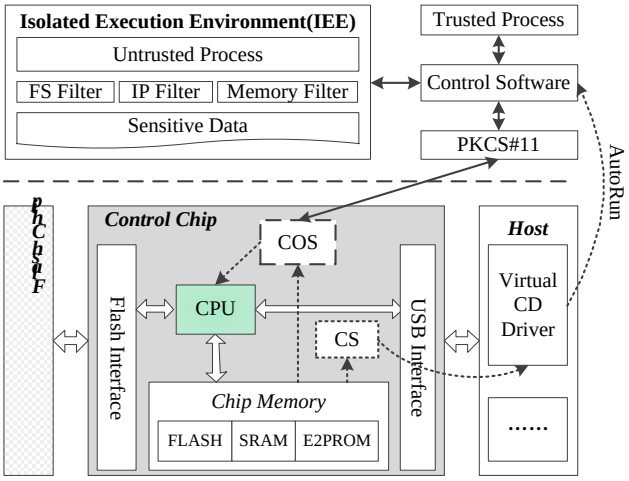

Fig. 1. The architecture of UTrustDisk.

2.2 Hash Trees

Hash trees also named Merkle trees were proposed in [7] as a protocol to authenticate data integrity efficiently. **Fig. 2** illustrates the structure of a 3-ary hash tree adopted in data integrity protection. A chunk means a variable-sized sequence of bytes, which is the unit for data validating. A typical hash tree is organized as an *m*-ary tree and the nodes and leafs are constructed into chunks. The leaf (named data chunk) in the tree contains the actual data, and the hash chunk (or the inner node) stores the collision resistant hash value of all the following *m* children-nodes. The hash result of the root chunk is stored in protected memory where it cannot be tampered with. All other chunks are kept in untrusted memory which is protected via hash trees.

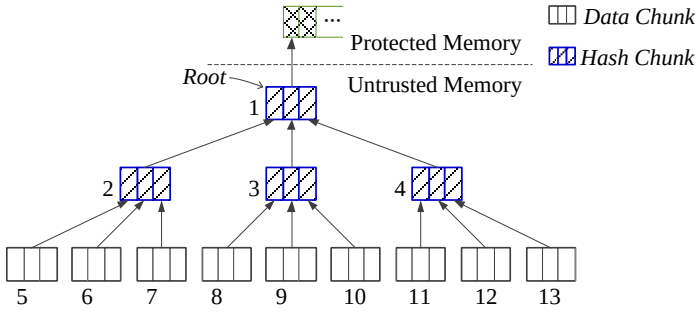

Fig. 2. Hash trees for data integrity protection. The memory is divided into equal length blocks: data chunks and hash chunks, both are resided in untrusted memory. The hash result of the root is kept in protected memory which cannot be tampered with.

To check the data's integrity of a data chunk or hash chunk, we need to 1) read the chunk and compute its hash value, 2) compare the hash result with the original hash stored in its parent chunk. These steps need to be repeated on the hash value recursively, until it reaches the value stored in protected memory. If any mismatch is detected in this process, the data integrity verification is failed. Similarly, to update the data in a chunk we need to check the integrity of the chunk and then update all the nodes from it to the root. With a balanced tree, the worst case time complexities of each read or write operation will be $O(\log_m(N))$ where N is the size of the memory, and the corresponding memory overhead is 1/(m-1) [18].

Merkle trees are widely used in storage and memory integrity protection. In [19], Maheshwari et. al build trusted databases on untrusted storage by using hash trees. Gassend et. al proposed a memory authentication scheme in [20] which was also based on Merkle trees, and their scheme improved the performance significantly by using caches. In [21], Clarke et. al proposed an offline verification scheme based on the incremental hashing [22], named multiset hash functions, instead of SHA-1. Hu, Hammouri and Sunar used universal hash function family NH to build a fast real-time memory authentication scheme [13], which is much faster compared to similar schemes based on SHA-1. The universal hashing is usually faster than others because it can generate the new hash incrementally [12][13].

2.3 Universal Hashing

Carter and Wegman proposed universal hashing in [23]. The main idea behind universal hashing is to select the hash function at random from a carefully designed class of functions at the beginning of execution. Using universal hashing can yield provably good performance and security.

Let H be a finite collection of hash functions that map a given finite set A with size a to a finite set B with size b. Let M represents the length of message string which is divided into m blocks with length w. For a given hash function $h \in H$ and a pair of distinct message pair $x, y \in A$, the following function is defined: $\delta_h(x, y) = 1$ if $h(x) = h(y)$, and $\delta_h(x, y) = 0$ otherwise. For a given set of hash functions H, $\delta_H(x, y)$ is defined as $\sum_{h \in H} \delta_h(x, y)$. In other words, $\delta_H(x, y)$ counts the number of functions in H for which x and y collide. When a hash function h is chosen randomly, the probability that two distinct inputs x and y yield a collision equals $\delta_H(x, y) / |H|$ [13]. We introduce the definitions of universal hash functions used in this paper from [24]: A finite collection of hash functions $H = h : A \to B$ is said to be universal if for every $x, y \in A$ where $x \neq y$, $|h \in H : h(x) = h(y)| = \delta_H(x, y) = |H| / b$.

Black et al introduced an almost universal hash function family called NH in [8]. But NH is not the best solution in applications deployed in ultra-low power devices. Kaps, Yuksel and Sunar proposed an energy scalable universal hashing named WH in [12]. In the same paper, the experimental results show that WH can obtain substantial power savings of up to 59% and a speedup of up to 7.4 times over NH.

3. Integrity Verification Algorithm

3.1 CP-ABE Algorithm

Hash trees are commonly used in data integrity protection as described in Section 2. However, the overhead of hash trees is too huge to be applied to USB flash disks. Taking the 3-ary Merkle tree in **Fig. 2** as an example, one third of the memory will be consumed by hash results. What's more, the additional data access operations caused by verifying hash trees may become the main bottleneck. Usually, the space overhead can be reduced vividly by increasing the fanout m of Merkle tree and using universal hashing such as WH. And the space consumption could be regarded as acceptable because the unprotected flash storage is very cheap. But the additional data verifying operations is really sure a performance killer. For a typical flash chip with size 4 GB, verifying a chunk may cause tens of read accesses.

In order to reduce the flash bandwidth pollution, we proposed a new scheme named Speculative caching and WH-hashing scheme (SWHash). In this scheme we adopt WH

universal hash function for chunks hashing. WH's hash result consumes half storage of NH, which reduces the space overhead greatly. The MAC is generated by WH hash results (we call it tags) of the chunk. All the memory will be partitioned into data chunks and hash chunks with the same size. Then the chunks can be processed by the same WH hash processing. Finally, the Merkle tree is built logically for protecting data integrity. Similar to other existing memory authentication protocols, we also use caches for reducing accesses. However, the cache in the USBKey processor is not large enough, and the hot chunk is often evicted by general chunks. So we propose the speculative caching to promote the existing cache policy

Speculative caching strategy's main idea is still hot chunk caching. In this scheme we occupy an amount of on-chip memory (named cache too) to store the root chunk of the sub hash tree which contains the hot chunk. Traditional cache policies work well when there are a lot of hot spots. The UTrustDisk is a flash disk which is used for storing and transferring files. Its access pattern is different from general applications. Because the data stored in the UTrustDisk is usually located in adjacent chunks, the read/write operation occurs on sequential chunks sequentially. So the hot spot is not obvious when reading file from UTrustDisk or writing to it. Since hot chunks are changing according to applications, we improve the cache policy by using speculative caching technique. We take the hash tree in **Fig. 2** as an example to illustrate the speculative strategy. Given a file stored in chunk 6, 7 13, and now we consider the read operation. Suppose chunk 2 is already cached in on-chip memory, when beginning to cache chunk 3, we can cache the chunk 4 speculatively at the same time. The speculative operation can reduce the additional memory accesses caused by integrity verification. To describe our scheme more clearly later, we define the chunk 3 as the left-neighbor of chunk 4 and the chunk 4 as the right-neighbor of chunk 3.

3.2 SWHash Algorithm

In the SWHash scheme, the memory is partitioned into chunks and a l-level Merkle tree is built up for integrity verification. In order to implement speculative caching, a counter C_x is added to every chunk M_x. The counter C_x records the number of M_x's left-neighbors. When C_x reaches the threshold, M_x's right-neighbor will be cached speculatively.The tag T_x is cached for reducing the redundant operations and pre-computing is adopted for speedup. We use the 32-bit WH universal hash function twice and then concatenate the hash values to obtain a 64 bits tag for each chunk. According to [12], the final hash result's collision probability is equal to 2^{-64}.

We outline the basic idea of SWHash protocol without the speculative caching and pre-computing in **Fig. 3**. And the full scheme of SWHash algorithm is presented by Algorithm 1 to Algorithm 5.

Algorithm 1: ***Initialization*** (called when the scheme is startup)

1. Initialize the cache.
2. For each chunk M_x in untrusted memory, calculate the hash value of each chunk M_x, and name it using the tag T_x. If the chunk M_x is the root (the topmost chunk), save T_x in protected memory; otherwise keep it in parental chunk.

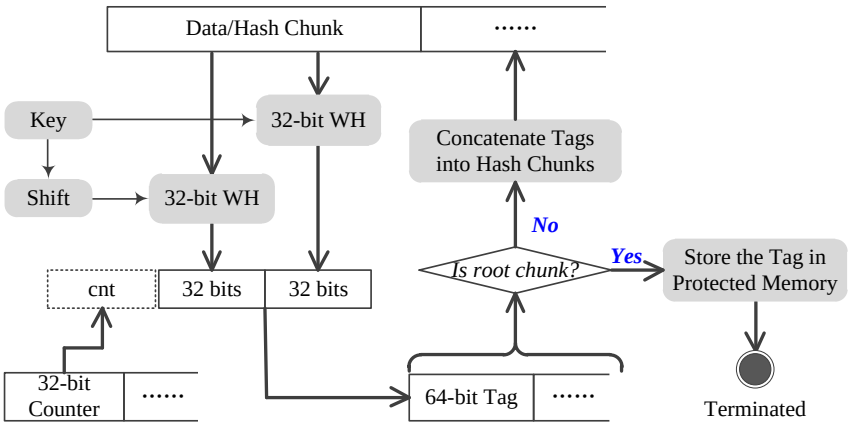

Fig. 3. Outline of the SWHash basic scheme. The data or hash chunk is hashed by 32-bit WH hash function twice to generate a tag. And the tags are concatenated and partitioned into chunks.

Algorithm 2: ***ReadAndCheck(x)*** (called when the processor reads data x in chunk M_x)

1. If the chunk M_x is cached, return the cached data x. If C_x reaches the threshold, and M_x's right-neighbor M_z exists, put M_z in cache and set $C_z = C_x + 1$. Then set $C_x = 0$. Operation completed.
2. Call ***ReadAndCheckChunk(M_x)***.
3. Put the chunk M_x into the cache. Return the data x.
4. Put the tag T_x in cache. Generate a counter C_x and set $C_x = 0$.
5. If M_x's left-neighbor M_y exists, set $C_x = C_y + 1$.

Algorithm 3: ***ReadAndCheckChunk(M_x)*** (called when reads chunk M_x and checks it)

1. Return M_x to the caller for speculative execution.
2. If M_x is the root, read T_x from protected memory; otherwise call ***ReadAndCheck(T_x)***.
3. Compute the hash value H_x of the chunk M_x. If $H_x \neq T_x$, the integrity check fails.
4. If the chunk M_x is the root, the integrity check succeed; otherwise, raise an exception.

Algorithm 4: ***Write(x)*** (called when the processor writes data x in chunk M_x)

1. If the chunk M_x is cached, modify the cached data x directly. If C_x reaches the threshold, and M_x's right-neighbor M_z exists, put M_z in cache and set $C_z = C_x + 1$. Then set $C_x = 0$. Operation completed.
2. Call ***ReadAndCheckChunk(M_x)***.
3. Put the chunk M_x and the tag T_x into the general cache. Generate a counter C_x and set $C_x = 0$. If M_x's left-neighbor M_y exists, set $C_x = C_y + 1$.
4. Modify the data x.

Algorithm 5: ***WriteBack(M_x)*** (called when the chunk M_x is evicted and it is dirty)

1. Write the chunk M_x to memory.
2. Read the tag T_x from cache.
3. Update the T_x incrementally to T'_x.
4. If the chunk M_x is the root, update T_x to T'_x in protected memory; otherwise call ***Write***(T'_x).

4. Data Leakage Prevention

As described before, data confidentiality is protected by UTrustDisk's encryption mechanism and we only store encrypted data in the flash chip. So UTrustDisk can prevent data leakage from unauthorized attacks. However, when the authorized user use UTrustDisk on an untrusted platform, the sensitive data will have to be decrypted and exposed to the host. So there is still a risk that some attackers may be able to sniff the private information.

4.1 Dynamic Isolation Requirements

The dynamic isolation is implemented based on light weight virtual machine. **Fig. 4** illustrates the dynamic isolation requirements. At the startup time, the trusted process P_1 is running under the monitor mode in IEE and P_2 running in untrusted host domain. There are three types of data leakage which need to be prevented by dynamic isolation as follows, and the corresponding solution is also listed out.

1. Untrusted process P_2 trying to read sensitive data d_1: P_2 will be migrated into IEE and renamed as P'_2. P'_2 is run under the monitor mode and authorized to access d_1.
2. Trusted process P_1 trying to write normal data d_2: d_2 will be copied into UTrustDisk and we rename it as d'_2. d'_2 is protected by UTrustDisk and no sensitive data will be leaked out from P_1.
3. Trusted process P_1 trying to send a message to Untrusted process P_2: P_2 will be migrated into IEE and run under the monitor mode.

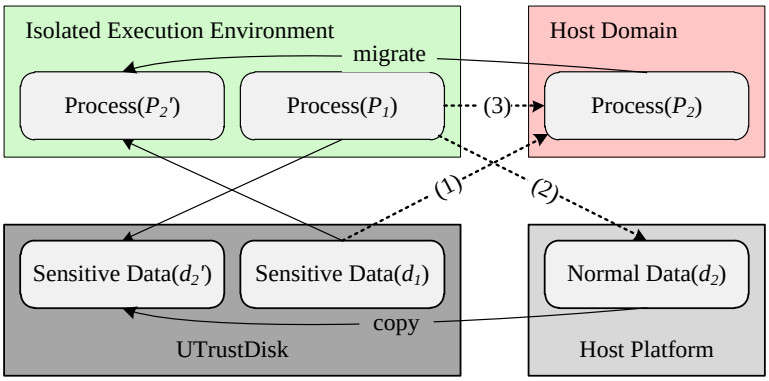

Fig. 4. Illustration of dynamic isolation requirements.

4.2 Data Leakage Prevention Scheme

In order to prevent information leakage in the usage stage, we implement dynamic isolation in IEE. IEE is an isolated execution environment based on TVDs, and processes running in IEE can be treated as trusted. We implement IEE based on FVM [25]. Most existing TVDs is built on hardware-level virtual machine systems, which costs lots of storage and computation capability [15][26]. In [27], W. Sun implemented an isolated file system which could redirect the sensitive data's operations to a security domain. And Y. Yu [25] expanded the isolation to registry, network, system call and so on.

UTrustDisk redirects file accesses to flash chip and isolates all related operations on registry, memory and networks. The file access redirection is carried out by file system filter driver, for example, we use minifilter for filtering unauthorized operations. Based on NDIS network filter and Detours APIs, UTrustDisk implements the network filter. Using the network filter, we can prevent data leak out by the malicious softwares.

Fig. 5 illustrates the data leakage prevention in UTrustDisk. At first the untrusted process is running in normal mode and it can access all the system resources such as storage, network and registry. Once the process accesses the sensitive data, it will be transformed into IEE and begin to run in the monitor mode. In the monitor mode, the process cannot access the normal resource any more, and all the related access will be redirect to UTrustDisk and other fake resources. The fake resources is built by TVDs, and the related data is stored inside UTrustDisk.

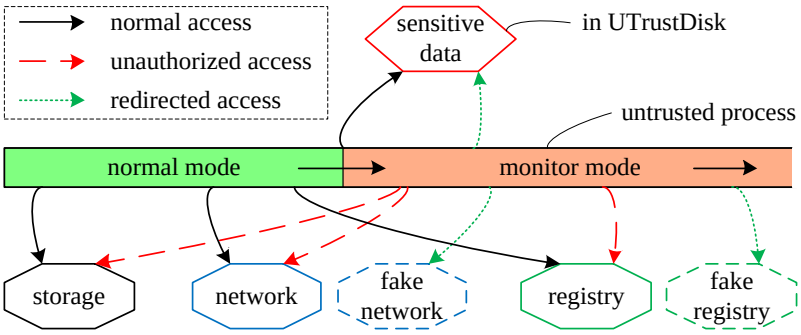

Fig. 5. Outline of data leakage prevention.

4.3 Security Analysis

The main purpose of the above scheme is to protect data from leakage. Thus we can describe the security through a question: could the data inside UTrustDisk be leaked out without going against the IEE. The security analysis is based on the model of secure information flow [28].

Let *FS* and *FO* represent the file sets inside and outside UTrustDisk, and let *PS* and *PO* be the collections of processes inside and outside IEE. And we use $a \rightarrow_t b$ to represent that there is a data flow from *a* to *b* at time *t*, *a* and *b* are files or processes. According to the definition of IEE, there are four rules in UTrustDisk.

Rule 1: For each process *p* inside IEE, *p* can not running outside until its termination. Therefore we have $\forall p \in_{t_0} PS, t \geq t_0$, there is no $p \in_t PO$.

Rule 2: For each process *p* outside IEE, once *p* access the file inside UTrustDisk, it will be turn to run inside IEE. That is $\forall p \in_{t_0} PO, \forall f \in_{t_0} FS, f \rightarrow_{t_0} p \Rightarrow p \in_t PS, t \geq t_o$.

Rule 3: For each process p inside IEE, p's access to system resources will be redirect to UTrustDisk, so $\forall p \in PS, \forall f \in FO, p \xrightarrow{try\ to\ access} f \Rightarrow fake(f')$ and $f' \in FS$, $p \to f'$.

Rule 4: For each process p outside IEE, once there is a data flow from p to p' and $p' \in PO$, p' will be turn to run inside IEE. So we have $\forall p \in_{t_0} PS, \forall p' \in_{t_0} PO, p \to_{t_0} p' \Rightarrow p' \in_t PS, t \geq t_0$.

Based on the rules above, we can introduce the security theorem of UTrustDisk's leakage prevention.

Theorem 1. For every file inside UTrustDisk, it will not be leak out. We can describe it formally as follows: $\forall f \in_{t_0} FS, \forall f' \in_{t_0} FO, t \geq t_0$, there is no $f \to_t f'$.

Proof.

We assume that this theorem is not established, $\forall f \in_{t_0} FS, \forall f' \in_{t_0} FO \Rightarrow \exists t \geq t_0$, $f \to_t f'$. Because all file accesses are operated by processes, we can present the data flow as follows. $\exists p_0, p_1, ..., p_n \in (PS \bigcup PO), t_0 \leq t_1 \leq ... t_n \leq t \Rightarrow f \to_{t_0} p_0, ..., p_n \to_{t_0} f', n \geq 0$.

According to Rule 2, $p_0 \in_{t_0} PS$, and according to Rule 4, we derive out that $p_1, ..., p_n \in_t PS$. As described in Rule 1, p_n will be inside PS, so we can conclude that $f' \in FS$ by Rule 3, which is conflicted with $f' \in FO$. So this theorem is established. ■

5. Performance Evaluation

We evaluate the performance of SWHash and data leakage prevention scheme by the UTrustDisk prototype. The UTrustDisk prototype contains an USBKey chip and a 4GB flash chip. The USBKey chip is produced by Nationz Technologies Company [6], and the product model is Z32H256D32U. The flash chip is produced by SAMSUNG and the product model is K9LBG08U0M [29].

All experiments were performed on a 2-core Intel CoreTM 2 T9500 (2.60 GHz) with 2 GB of memory (667 MHz) running Windows 7 Ultimate (Service Pack 1). UTrustDisk's main parameters are listed in **Table 1**. There is a 32KB EEPROM in USBKey chip. And the block size of write/read operation in COS is 512 bytes, so the chunk size is set to 512 bytes too.

Table 1. The main parameters of utrustdisk prototype.

Symbol	Value	Description
N	31	number of hash trees
L	4	number of hash trees' level
M	4 GB	size of untrusted flash chip
T	8	bytes size of tag
B	512	bytes size of data/hash chunk

The experiment tool is ATTO Disk Benchmark (version 2.46) [30]. The ATTO Disk Benchmark performance measurement tool can provide the highest level of performance to UTrustDisk with various transfer sizes and test lengths for reads and writes. In our experiments the related options are customized as follows: the transfer sizes are set from 0.5 KB to 8 MB; the transfer length is 1 GB; the queue depth is 4 and the overlapped I/O is selected.

In order to evaluate the optimization result of speculative caching and pre-computing, we turn off them to form a basic scheme. We test the read/write performance of the basic scheme and SWHash scheme. The experimental results are shown in **Fig. 6** and **Fig. 7**. The basic and SWHash scheme are compared with naive accesses, which name none in the figure. Due to speculative caching and pre-computing improvements, SWHash increases the read and write speed up to 13% and 28% over basic scheme. However, when the data size is below 2 KB, the SWHash consumes more time than the basic scheme due to inefficient speculation. SWHash's average write performance penalty is 13.76% and the average read overhead is 10.18%.

We also test the SHA-1 scheme and NH scheme as contrast to evaluate SWHash's efficiency. The SHA-1 scheme is described in [31] and the cache is also employed for speed. The NH scheme is presented in [12] and the cache is also adopted. **Fig. 8** presents the write speed comparison among them. Because the conclusion derived from read experiments is similar to write, we only present the write results. This experiment shows that SWHash is much better than SHA and NH scheme. When compared to a NH scheme, our prototype-based results show that SWHash can increase write throughput up to 52%. When compared to SHA-1 scheme, it increases write speed by up to 248%-350%.

To evaluate the speedup of speculative caching, we record the success rate of speculation. A success speculation means a chunk is selected to cache speculatively and it has been accessed before evicted. So the success rate is the proportion of the mount of success speculation to all the cache chunks. **Fig. 9** illustrates the results. When the data transfer size reaches 8 KB, the speculation speed up significantly. And the success rate of speculative caching reaches 94.5%.

We also evaluate the overhead of data leakage prevention scheme. We turn off the SWHash scheme in this evaluation. The experimental results are shown in **Fig. 10**. The none model means turning off the data leakage prevention mechanism and protected model means using it. The average write penalty caused by the data leakage prevention scheme is about 15.31%. The main reason of this penalty is that the implementation of IEE causes additional overhead such as processes migration.

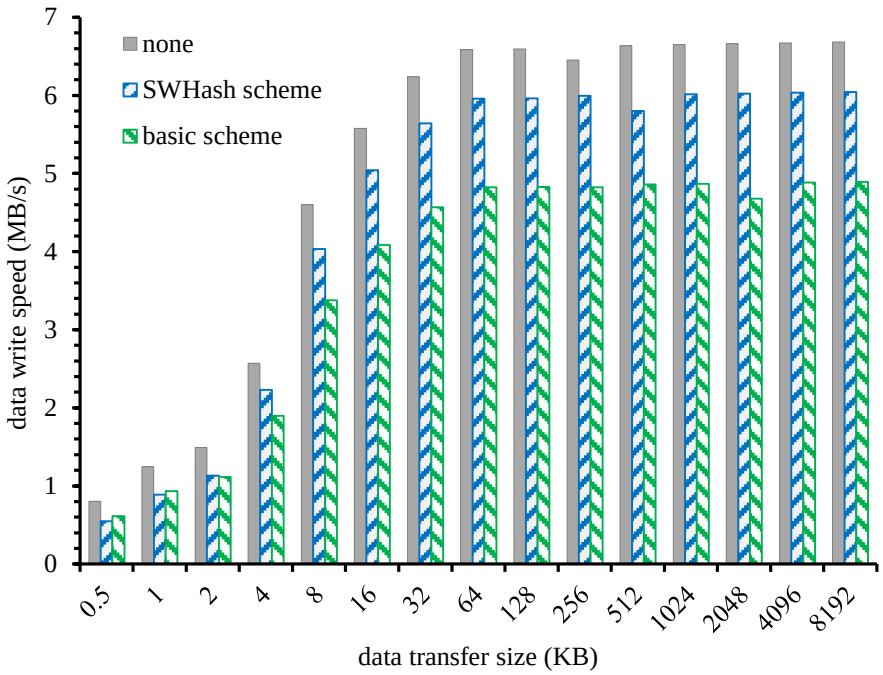

Fig. 6. The write performance of the basic scheme and SWHash scheme.

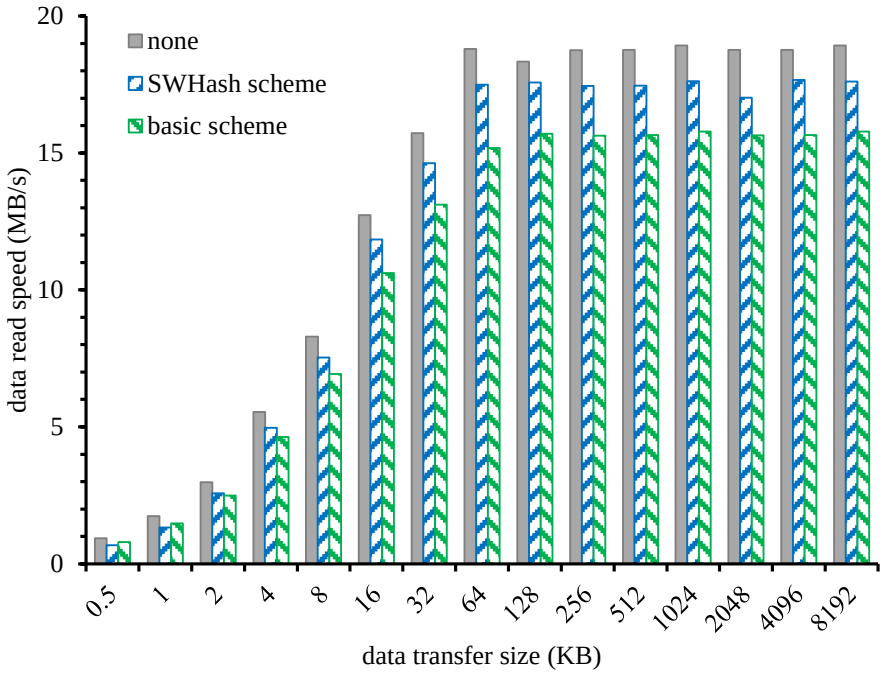

Fig. 7. The read performance of the basic scheme and SWHash scheme.

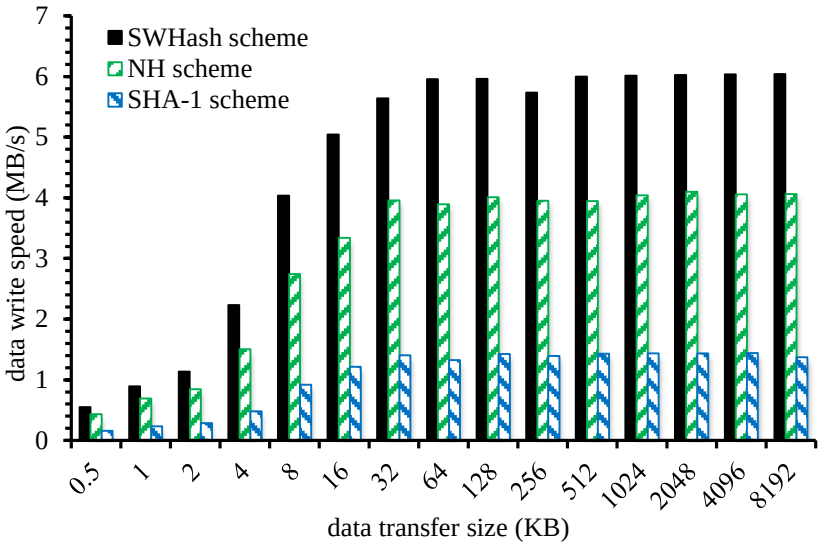

Fig. 8. The write speed comparison among SHA-1, NH and SWHash.

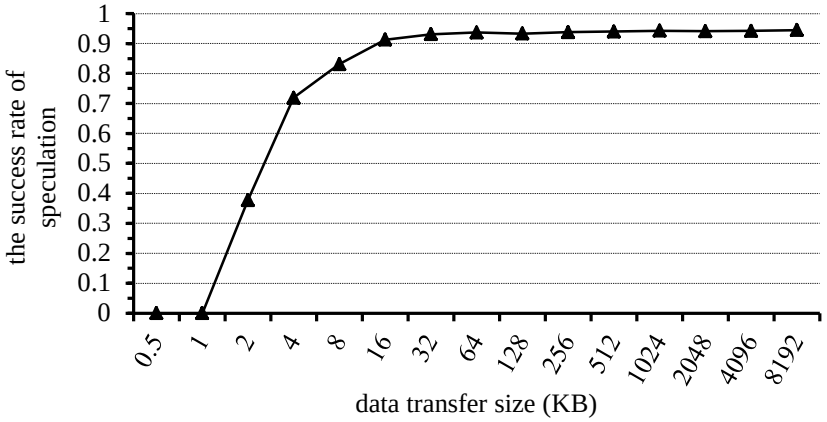

Fig. 9. The success rate of speculation.

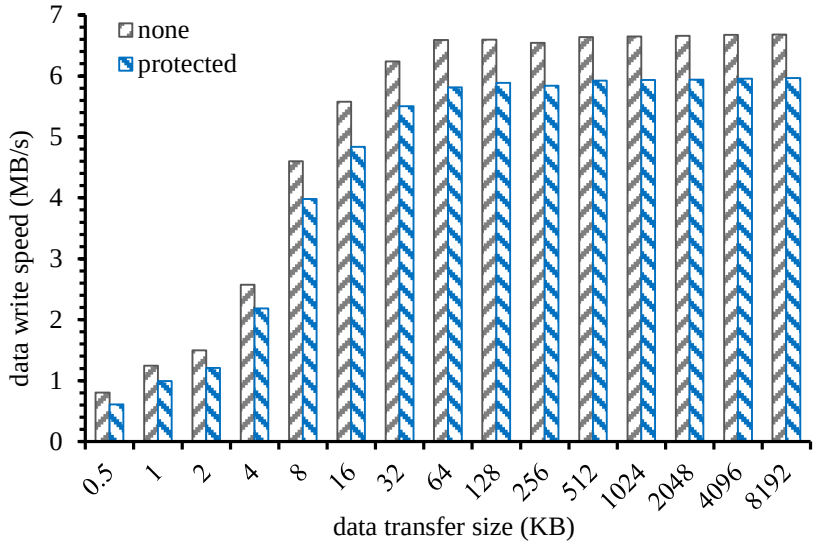

Fig. 10. The overhead of data leakage prevention scheme.

6. Conclusion

In this paper, we proposed UTrustDisk for protecting the data security in flash. The advantages of UTrustDisk are obvious: First, the security of data confidentiality and leakage prevention can be proved; Second, the mechanism of integrity verification is optimized, achieving a great speedup over the SHA-1 and NH scheme.

In the future we will optimize our scheme by parallelism. Nowadays the multi-core processor is wild used, if the integrity verification and data encryption operations can be allocated to cores evenly, we will achieve a significant speedup.

References

[1] Computer Security Institute, 16th annual CSI computer crime and security survey executive summary, http://www.gocsi.com, 2011. Article (CrossRef Link)

[2] Trusted Computing Group, TCG storage architecture core specification, http://www.trustedcomputinggroup.org, 2011. Article (CrossRef Link)

[3] F. Hou, D. Gu, N. Xiao, and Y. Tang, "Data privacy and integrity appropriate for disk protection," in *Proc. of the 8th IEEE Int. Conf. on Computer and Information Technology*, pp.414-419, July 8-11, 2008. Article (CrossRef Link)

[4] Y. Cheng, Z. Wang, J. Wu, S. Mei, J. Ren, and J. Ma, "SWHash: An Efficient Data Integrity Verification Scheme Appropriate for USB Flash Disk," in *Proc. of the 10th Int. Conf. on Trust, Security and Privacy in Computing and Communications*, pp.381-388, November 16-18, 2010. Article (CrossRef Link)

[5] J. Ma, Z. Wang, J. Ren, C. Liu, J. Wu, Y. Cheng, and S. Mei, "Trsf: Implementing active removable storage protection via trusted virtual domains," *Chinese Journal of Electronics*, vol. 40, no. 2, pp.376-383, February, 2011. Article (CrossRef Link)

[6] Nationz Technologies Company, Secure storage chips, http://www.nationz.com.cn/Products2.aspx?id=36, 2011. Article (CrossRef Link)

[7] R.C. Merkle, "Protocols for public key cryptosystems," in *Proc. of the 1980 IEEE Symposium on Security and privacy*, pp.122-134, April, 1980. Article (CrossRef Link)

[8] J. Black, S. Halevi, H. Krawczyk, T. Krovetz, and P. Rogaway, "Umac: Fast and secure message authentication," in *Proc. of the 19th Annual International Cryptology Conference (CRYPTO'99)*, pp.79–79, August 15-19, 1999. Article (CrossRef Link)

[9] R. Huang and G.E. Suh, "Ivec: off-chip memory integrity protection for both security and reliability," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, pp.395-406, June, 2010. Article (CrossRef Link)

[10] W. Shi, H.H.S. Lee, M. Ghosh, and C. Lu, "Architectural support for high speed protection of memory integrity and confidentiality in multiprocessor systems," in *Proc. of the 13th Int. Conf. on Parallel Architectures and Compilation Techniques*, pp.123-134, September 29-October 3, 2004. Article (CrossRef Link)

[11] C. Yan, D. Englender, M. Prvulovic, B. Rogers, and Y. Solihin, "Improving cost, performance, and security of memory encryption and authentication," *ACM SIGARCH Computer Architecture News*, vol. 34, no. 2, pp.179-190, May, 2006. Article (CrossRef Link)

[12] J.P. Kaps, K. Yuksel, and B. Sunar, "Energy scalable universal hashing," *IEEE Transactions on Computers*, vol. 54, no. 12, pp.1484-1495, December, 2005. Article (CrossRef Link)

[13] Y. Hu, G. Hammouri, and B. Sunar, "A fast real-time memory authentication protocol," in *Proc. of the 3rd ACM workshop on Scalable trusted computing*, pp.31-40, October 27-31, 2008. Article (CrossRef Link)

[14] J.L. Griffin, T. Jaeger, R. Perez, R. Sailer, and L. Van Doorn, "Trusted virtual domains: Toward secure distributed services," in *Proc. of the 1st IEEE Workshop on Hot Topics in System Dependability*, June 28-July 1, 2005. Article (CrossRef Link)

[15] I. Burdonov, A. Kosachev, and P. Iakovenko, "Virtualization-based separation of privilege: working with sensitive data in untrusted environment," in *Proc. of the 1st EuroSys Workshop on Virtualization Technology for Dependable Systems*, pp.1-6, April 1-3, 2009. Article (CrossRef Link)

[16] Y. Yu, "OS-level virtualization and its applications," PhD thesis, State University of New York At Stony Brook, 2009. Article (CrossRef Link)

[17] Nationz Technologies Company, Armordisk (security usbkey) encrypted storage," http://www.nationz.com.cn/Solutions2.aspx?id=4, 2011. Article (CrossRef Link)

[18] D.Williams and E.G. Sirer, "Optimal parameter selection for efficient memory integrity verification using merkle hash trees," in *Proc. of the 3rd IEEE Int. Symposium on Network Computing and Applications*, pp.383-388, August 30- September 1, 2004. Article (CrossRef Link)

[19] U. Maheshwari, R. Vingralek, and W. Shapiro, "How to build a trusted database system on untrusted storage," in *Proc. of the 4th Conf. on Symposium on Operating System Design & Implementation*, pp.1-10, October 22-25, 2000. Article (CrossRef Link)
[20] B. Gassend, G.E. Suh, D. Clarke, M. Van Dijk, and S. Devadas, "Caches and hash trees for efficient memory integrity verification," in *Proc. of the Ninth Int. Symposium on High-Performance Computer Architecture*, pp.295-306, February 8-12, 2003. Article (CrossRef Link)
[21] Y. Hu and B. Sunar, "An improved memory integrity protection," in *Proc. of the 3rd Int. Conf. on Trust and Trustworthy Computing*, June, 2010. Article (CrossRef Link)
[22] M. Bellare, O. Goldreich, and S. Goldwasser, "Incremental cryptography: The case of hashing and signing," in *Proc. of the 14th Annual International Cryptology Conference (CRYPTO'94)*, pp. 216–233, August 21-25, 1994. Article (CrossRef Link)
[23] J.L. Carter and M.N. Wegman, "Universal classes of hash functions," *Journal of computer and system sciences*, vol. 18, no. 2, pp.143-154, 1979. Article (CrossRef Link)
[24] W. Nevelsteen and B. Preneel, "Software performance of universal hash functions," in *Proc. of the 1999 Int. Conf. on the Theory and Application of Cryptographic Techniques (EUROCRYPT'99)*, pp.24-41, May 2-6, 1999. Article (CrossRef Link)
[25] Y. Yu, F. Guo, S. Nanda, L. Lam, and T. Chiueh, "A feather-weight virtual machine for windows applications," in *Proc. of the 2nd Int. Conf. on Virtual Execution Environments*, pp.24-34, June 14-16, 2006. Article (CrossRef Link)
[26] L. Catuogno, H. Lohr, M. Manulis, A.R. Sadeghi, and M. Winandy, "Transparent mobile storage protection in trusted virtual domains," in *Proc. of the 23rd Conf. on Large Installation System Administration*, pp.1-12, November 1-6, 2009. Article (CrossRef Link)
[27] W. Sun, Z. Liang, R. Sekar, and VN Venkatakrishnan, "One-way isolation: An effective approach for realizing safe execution environments," in *Proc. of the 2005 Network and Distributed Systems Symposium*, February 3-4, 2005. Article (CrossRef Link)
[28] D.E. Denning, "A lattice model of secure information flow," *Communications of the ACM*, vol. 19, no. 5, pp.236-243, May, 1976. Article (CrossRef Link)
[29] Samsung, "Products flash," http://www.samsung.com/global/business/semiconductor/products/flash/Products/Flash.html, 2011. Article (CrossRef Link)
[30] ATTO Technology Inc, "Disk benchmark," http://www.attotech.com/products, 2011. Article (CrossRef Link)
[31] M. Etzel, S. Patel, and Z. Ramzan, "Square hash: Fast message authentication via optimized universal hash functions," in *Proc. of the 19th Annual International Cryptology Conference (CRYPTO'99)*, pp.786-786, August 15-19, 1999. Article (CrossRef Link)

Yong Cheng is an an engineer fellow at National University of Defense Technology, Chang'sha, P. R. China. He received his PhD degree in Computer Science and Technology from National University of Defense Technology. His current research interests include the storage security in public cloud services, trusted computing in public cloud or other un-trusted devices, cryptographic algorithm design for data security, data leakage prevention, etc.

Jun Ma is an assistant professor at National University of Defense Technology, Chang'sha, Hu'nan, P.R.China. He received his PhD degree in Computer Science and Technology from National University of Defense Technology. His current research interests include the security of operating system and cloud storage, trusted computing and data security, etc.

Jiangchun Ren is an associate professor in National University of Defense Technology, Changsha , P.R.China. He received his PHD degree in computer science and technology from National University of Defense Technology. His current research interests mainly include trusted computing, storage systems (in particular the cloud storage), data management and computer architecture (in particular the trusted computing).

Songzhu Mei is an assistant research fellow at National University of Defense Technology, Chang'cha P. R. China. He received his PhD degree in Computer Science and Technology from National University of Defense Technology. His current research interests include the trusted computing, distributed system, cloud computing, etc.

Zhiying Wang is a professor at National University of Defense Technology, Chang'cha P. R. China. He received his PhD degree in Computer Science and Technology from National University of Defense Technology. His current research interests include information system security, advanced computer architecture, microprocessor design, asynchronous architecture design technology, etc.