# A Non-repudiable Dynamic Provable Data Possession

Jun-Feng Tian[1,2], Rui-Fang Guo[1,2(✉)], and Xuan Jing[1,2]

[1] School of Cyberspace Security and Computer Institute, Hebei University, Baoding 071000, China
grf.skzxc@gmail.com
[2] Hebei Key Laboratory of High Confidence Information Systems, Hebei University, Baoding 071000, China

**Abstract.** With the widespread popularity of cloud storage, cloud storage security issues have also received much attention. A provable data possession (PDP) scheme can effectively help users to verify the integrity of data stored remotely in the cloud. For this reason, the client's PDP scheme is constantly improving and developing. In view of the problem that the existing PDP scheme pays less attention to the clients deceiving the cloud server, a non-repudiable dynamic PDP scheme based on the Stern-Brocot tree (SB-NR-DPDP) is proposed. We put forward a dynamic storage structure and dynamic operation algorithm based on the Stern-Brocot tree, so that it can satisfy the client's dynamic data operations and realize the non-repudiation feature of the scheme. This scheme can resist hash value attacks, delete-insert attacks and tamper with cloud return value attacks. The theoretical analysis shows that the proposed scheme has less computing and storage overhead than other schemes.

**Keywords:** Cloud storage · Provable data possession · Stern-brocot · Dynamic operation

## 1 Introduction

As cloud storage can provide users with high-quality data storage and computing services [1], cloud storage has gradually gained wide popularity among users. Cloud storage not only provides convenience for users but also raises serious security problems for them. Cloud storage not only makes users relinquish physical control of the data but also increases the risk of data being leaked by, tampered with and deleted by cloud service providers. In addition, the security of cloud storage is threatened by external attackers, hardware failures and other factors. Therefore, research on the integrity verification of users' cloud data is urgently needed.

A provable data possession (PDP) scheme can effectively help users to verify the integrity of data stored remotely in the cloud. However, research on PDP has paid little attention to user deception by cloud service providers. For example, a user once issued an order to delete a certain piece of data to the cloud service provider, but the user denied that order when authenticating the integrity and blamed the cloud service provider, resulting in disputes between the user and the cloud service provider. For this


J. Li et al. (Eds.): SPNCE 2019, LNICST 284, pp. 723–730, 2019.
https://doi.org/10.1007/978-3-030-21373-2_61

reason, by introducing the Stern-Brocot tree type of dynamic data structure, this paper proposes a non-repudiable dynamic provable data possession scheme (SB-NR-DPDP).

## 2 Related Work

To solve the problem of users checking cloud data integrity, researchers have proposed a provable data possession (PDP) scheme. In 2007, Ateniese et al. [2] first proposed the PDP scheme. To support user dynamic operations and to improve the scheme's flexibility of the scheme, the researchers proposed dynamic PDP scheme. For example, [1, 3, 4]. To eliminate complex key and certificate management and to improve PDP scheme efficiency, Zhao et al. [5] proposed the first identity-based PDP scheme in 2013. To solve the problem of user unreliability and improve the credibility of both parties in a PDP scheme, in 2014, Mo et al. [6] proposed a non-repudiation PDP scheme based on the Merkle hash tree and timestamps. Feng [7] et al. found that the existing dynamic data structure could not satisfy the non-repudiation feature of the PDP scheme very well. By introducing a logical index table (ILT), they proposed the non-repudiation and identity-based, non-repudiable dynamic PDP scheme (ID-NR-DPDP) in cloud storage.

## 3 The System Model

An SB-NR-DPDP scheme model contains four primary entities: the private key generator (PKG), the data owner (User), the cloud server provider (CSP), and the unbiased judge. As shown in Fig. 1, their functions are as follows.

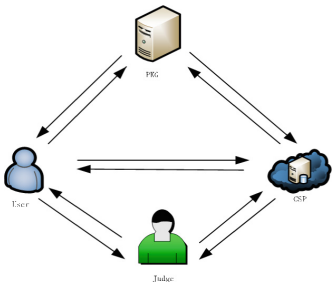


**Fig. 1.** SB-NR-DPDP scheme model

- PKG: A trusted third party, which is called the private key generator. It can help users generate private keys.
- User: The data owner who uses cloud storage services to outsource data to remote clouds.

- Cloud server: A semi-trusted entity that stores and processes the user's data. It can prove data integrity to clients, but sometimes the cloud server can destroy data integrity and trick clients into believing that the data are still intact in the cloud.
- Judge: A trusted third party that resolves disputes when they arise between the user and the cloud service provider.

## 4 The Dynamic Operation Algorithm of the Stern-Brocot Tree

The Stern-Brocot tree [8] is a binary tree used to construct a set consisting of all non-negative minimal fractions. It was discovered independently by German mathematician Moritz Stern and French watchmaker Achille Brocot. The dynamic operation algorithm in the Stern-Brocot tree includes an insert, delete and modify algorithm. Users and the cloud server initialize the tree: the root node is $\frac{1}{1}$ and is used to form a tree structure that is symmetric to the root node. The left child of the root node is $\frac{N}{M}$, which is the seed node. The right child is $\frac{M}{N}$. Using a seed, a Stern-Brocot tree with a symmetric root node can be established. As shown in Fig. 2, all the fractions in the tree are in the simplest form.

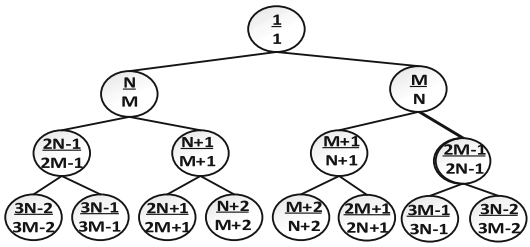


**Fig. 2.** Partial Stern-Brocot tree with seed (N, M)

The algorithm calculates the height of the tree that needs to be established according to the number of seeds and the number of data blocks n. Each leaf node in the tree corresponds to a unique pointer variable, and each pointer variable points to the user's corresponding data block $F_{wx}$.

Insert algorithm: When data blocks $F_{wx}$ need to be inserted, an update from the most recent operation starts after the largest leaf node. The pointer variable wx corresponding to the appropriate insert block position is found. It then points to the file block $F'_{wx}$. The number of blocks is updated at the same time.

Deletion algorithm: To delete the correspondence between pointer variables and file blocks, it needs to delete the wx pointer to $F_{wx}$, and let wx = 1 as failure node that is to add wx as a global pointer variable for the dynamic operation algorithm. Finally, update the number of blocks at the same time, n = n − 1. The purpose of marking the failure node is that when the tree is built again, wx conflicts with the global variable

value, indicating that the position is the failure position. Continue to look for the insertion position, so as to prevent the deletion - insertion attack. If the deleted position is at the last block, a new block will be inserted after the newly deleted position to avoid the delete - insert attack. When there are not enough leaves in the tree, a row will be generated again and the global variable will be cleared after initialization.

Modify algorithm: First, it removes the relationship between the pointer variable wx and the file block $F_{wx}$, and makes pointer variable value wx = 1. It then adds wx as a global pointer variable for the dynamic operation algorithm. Then, it follows the insertion algorithm to find pointer variable wx′ corresponding to the insertion location, and points to modify the file.

## 5 Details of the SB-NR-DPDP Scheme

The SB-NR-DPDP scheme include six algorithms: Setup, Extraction, Tagging, Processing, Proof, and Judgement, which are described in detail in the following sections.

### 5.1 Setup

This algorithm is executed by the PKG. Let $G_1, G_2$ be a cyclic multiplication groups with prime order, and g be a generator of $G_1$. The map $e : G_1 \times G_1 \rightarrow G_2$ is a bilinear pairing. The PKG defines three hash functions: $H : \{0,1\}^* \rightarrow G_1$, $h : \{0,1\}^* \rightarrow Z_q^*$, $h_1 : \{0,1\}^* \rightarrow Z_q^*$; a pseudo-random function: $\varnothing_{key} : key \times \{0,1\}^* \rightarrow Z_q^*$; and a pseudo-random permutation: $\pi_{key} : key \times \{0,1\}^{\log(\theta)} \rightarrow \{0,1\}^{\log(\theta)}$. The PKG then selects a random number $c \in Z_q^*$ and computes $C = c \cdot g$. The identity-based signature algorithm of Galindo and Garcia [9] where $\text{sign}(sk_{ID}, f) \rightarrow \ell$ generates the signature for the message, and $\text{verify}(ID, f, \ell)$ is used to verify the signature validity. PKG publishes the public parameters $G_g = \{G_1, G_2, q, g, e, H, h, h_1, C, \pi, \phi, sign(), verify()\}$ and keeps the msk = c secret.

### 5.2 Extraction

The PKG selects a random number $j \in_R Z_q^*$ and computes $R = j \cdot g$, $Z = j + c \cdot h(ID||R) \bmod q$; therefore, $sk_{ID} = (R, Z)$. The PKG uses this algorithm to generate the client's secret key, $sk_c$, or the cloud server's secret key, $sk_s$.

### 5.3 Tagging

Given an F, the client chooses a random file name NI from some large domain and splits the file into n blocks, $F = F_1 \| F_2 \| \ldots F_n$, given $F_x$, $1 \le x \le n$, $F_x = F_{x1} || F_{x2} || \ldots F_{xs}$. Then, the client selects s random values $u_1, u_2, \ldots u_s \in G_1$, $U = (u_1, u_2, \ldots u_s)$, and then initializes the tree by the dynamic operation algorithm of the Stern-Brocot tree to obtain wx. It then calculates the signature $\ell_c$ and label $T_{wx}$, where $\ell_c = sign(sk_c, NI||U||n||(N, M))$ and $T_{wx} = Z_c \cdot \left( H(NI||wx||U) + \sum_{K=1}^{S} F_{wx,k} \cdot u_k \right)$. Then, the client uploads

$\{F_{wx}, T_{wx}, \mathrm{NI}, \mathrm{U}, \mathrm{n}, (\mathrm{N},\mathrm{M})\}$ to the cloud server. Next, the cloud server uses equations $\mathrm{e}\left(\sum_{x=1}^{n} T_{wx}, \mathrm{g}\right) = \mathrm{e}(\sum_{x=1}^{n} H(NI||wx||U) + \sum_{K=1}^{s}\left(\sum_{x=1}^{n} F_{wx}\right) \cdot u_k, R_c + \mathrm{h}(ID_c||R_c) \cdot \mathrm{C})$ and $1 = \mathrm{verify}(ID_c, \mathrm{NI}||\mathrm{U}||\mathrm{n}||(\mathrm{N},\mathrm{M}), \ell_C)$ to check the validity of $T_{wx}$, $(1 \le \mathrm{x} \le \mathrm{n})$ and $\ell_c$. If one of them does not hold, the operation stops. Otherwise, the cloud server stores them, computes the signature $\ell_s = sign(ID_s, \ell_c)$, and returns $\ell_s$ as a receipt to the client. The client receives the receipt from the cloud server, and then checks the validity of the receipt $\ell_s$ by using the equation $1 = \mathrm{verify}(ID_s, \ell_s, \ell_c)$. If it is invalid, the operation stops. Otherwise, the client stores $\{\mathrm{NI}, \mathrm{n}, (\mathrm{N},\mathrm{M}), \ell_s, \ell_c\}$ and deletes data blocks and tags from local storage.

### 5.4 Processing

- **Insert**

The client wants to insert a new block $\mathrm{F}'$. First, the client obtains the pointer variable $\mathrm{wx}'$ corresponding to the new insertion location and the number of file blocks n by using the dynamic operation algorithm. Then, the file $\mathrm{F}'$ is divided into s sections, $\mathrm{F}' = \left(F'_1 \middle\| F'_2 \middle\| \ldots F'_s\right)$. The client computes the new label $T'_{wx}$ and signature $\ell'_c$, by using the equations $T'_{wx} = Z_c \cdot \left(H(NI||wx'||U) + \sum_{k=1}^{s} F'_k \cdot u_k\right)$ and $\ell'_c = sign(sk_c, IN||NI||wx'||n)$. Next, the client uploads $\left\{\mathrm{IN}, \mathrm{F}', T'_{wx}, \mathrm{U}, \mathrm{n}, \ell'_c\right\}$, to the cloud server. Then, the cloud server checks the validity $T'_{wx}$ and $\ell'_c$, by using the equations $1 = \mathrm{verify}(ID_c, \mathrm{IN}||\mathrm{U}||\mathrm{wx}'||\mathrm{n}, \ell'_c)$ and $\mathrm{e}\left(T'_{wx}, \mathrm{g}\right) = \mathrm{e}(\mathrm{H}(\mathrm{NI}||\mathrm{wx}'||\mathrm{U}) + \sum_{k=1}^{s} F'_k \cdot u_k, R_c + \mathrm{h}(ID_C||R_c) \cdot \mathrm{C})$. If one of them does not hold, the operation stops; otherwise, the cloud server updates n, $\ell'_c$. Then, the signature $\ell'_s = sign\left(ID_s, \ell'_c\right)$ is computed and $\ell'_s$ is returned as a receipt to the client. The client receives the receipt from the cloud server and then checks the validity of the receipt $\ell'_s$ by using the equation $1 = \mathrm{verify}\left(ID_s, \ell'_s, \ell'_c\right)$. If it is invalid, the operation stops; otherwise, the client updates n, $\ell'_c$, $\ell'_s$, and deletes $\mathrm{F}'$, $T'_{wx}$ from local storage.

- **Delete**

The client wants to delete block $F_{wx}$. First, the client obtains the pointer variable wx corresponding to the delete location, and the number of file blocks n by using the dynamic operation algorithm. The client computes the signature $\ell'_c$, by using the equation $\ell'_c = sign(sk_c, NI||U||wx||n)$. Then, the client uploads $\left\{\mathrm{NI}, \mathrm{U}, \mathrm{n}, \mathrm{wx}, \ell'_c\right\}$ to the cloud server. Next, the cloud server checks the validity by using the equation $1 = \mathrm{verify}\left(ID_c, \mathrm{NI}||\mathrm{U}||\mathrm{wx}||\mathrm{n}, \ell'_c\right)$. If it holds, the cloud server updates n, $\ell'_c$. Then, the signature $\ell'_s = sign\left(ID_s, \ell'_c\right)$ is computed and $\ell'_s$ is returned as a receipt to the client. The client receives the receipt from the cloud server and then checks the validity of the receipt $\ell'_s$ by using the equation $1 = \mathrm{verify}\left(ID_s, \ell'_s, \ell'_c\right)$. If it is invalid, the operation stops; otherwise, the client updates n, $\ell'_c, \ell'_s$.

- **Modify**

The client wants to modify the file block value into F′. First, the client obtains the pointer variable wx′ corresponding to the new insertion location of the new block by using the dynamic operation algorithm. Then, the file F′ is divided into s sections - $\mathrm{F}' = \left(F_1' \middle|\middle| F_2' \middle|\middle| \ldots \middle|\middle| F_S'\right)$. The client computes the new label $T'_{wx}$ and signature $\ell'_c$, by using the equations $T'_{wx} = Z_c \cdot \left(H(NI||wx'||U) + \sum_{k=1}^{s} F'_k \cdot u_k\right)$ and $\ell'_c = sign(sk_c, NI||U||wx'||n)$. Then, the client uploads $\left\{\mathrm{F}', T'_{wx}, NI, U, wx', n, \ell'_c\right\}$ to the cloud server. The cloud server checks the validity $T'_{wx}$ and $\ell'_c$, by using the equations $1 = \mathrm{verify}(ID_c, \mathrm{NI}||\mathrm{U}||\mathrm{wx}'||\mathrm{n}, \ell'_c)$ and $\mathrm{e}\left(T'_{wx}, \mathrm{g}\right) = \mathrm{e}\left(\mathrm{H}(\mathrm{NI}||\mathrm{wx}'||\mathrm{U}) + \sum_{k=1}^{s} F'_k \cdot u_k, R_c + h(ID_c||R_C) \cdot C\right)$. If one of them does not hold, the operation stops; otherwise, the cloud server updates $\ell'_c$. Then, the signature $\ell'_s = sign\left(ID_s, \ell'_c\right)$ is computed and $\ell'_s$ is returned as a receipt to the client. The client receives the receipt from the cloud server and checks the validity of the receipt $\ell'_s$ by using the equation $1 = \mathrm{verify}\left(ID_s, \ell'_s, \ell'_c\right)$. If it is invalid, the operation stops; otherwise, the client updates $\ell'_c$, $\ell'_s$, and deletes the file block and its labels.

### 5.5 Proof

The client wants to verify the integrity of the file NI. First, the client selects a random number i, where $1 \le \mathrm{i} \le \mathrm{n}$ and $s_1, s_2 \in_R Z_q^*$. The client computes $S_2 = s_2 \cdot g, \hat{\ell}_c = sign(sk_c, NI||s_1||S_2||i)$ and sends the challenge $\mathrm{chal} = \left(\mathrm{i}, s_1, S_2, \mathrm{NI}, \hat{\ell}_c\right)$ to the cloud server. Upon receiving the challenge, the cloud server stops the dynamic operations of this file, selects $s_3 \in_R Z_q^*$, computes $S_3 = s_3 \cdot g$, $\mathrm{S} = s_3 \cdot S_2$, $\mathrm{Y} = \{\mathrm{y}(\pi_{s_1}(\rho))|1 \le \rho \le \mathrm{i}\}$, $a_y = \phi_S(y)$, $\mathrm{y} \in \mathrm{Y}$, $\hat{F}_k = \sum_y a_y F_{yk}$, $1 \le \mathrm{k} \le \mathrm{s}$, $\mathrm{T} = \sum_{y \in Y} a_y \cdot T_y$, $\hat{\ell}_s = sign(sk_s, S_3||n||(N, M)||\hat{F}_1||\hat{F}_2||\ldots||\hat{F}_S||\hat{T})$, and sends $\mathrm{re} = \left\{S_3, \mathrm{n}, (\mathrm{N}, \mathrm{M}), \hat{F}_1, \hat{F}_2, \ldots \hat{F}_s, \hat{T}, \hat{\ell}_s\right\}$ to the client as a response. Then, the client computes $\mathrm{S} = s_2 \cdot S_3$, $\mathrm{Y} = \{\mathrm{y}(\pi_{s_1}(\rho))|1 \le \rho \le \mathrm{i}\}$, $a_y = \phi_S(y)$, and $\mathrm{y} \in \mathrm{Y}$, and then checks the validity of response by using the equations $\mathrm{e}\left(\hat{T}, \mathrm{g}\right) = \mathrm{e}\left(\sum_{y \in Y} a_y \cdot \mathrm{H}(\mathrm{NI}||\mathrm{y}||\mathrm{U}) + \sum_{k=1}^{s} \hat{F}_k \cdot u_k, R_c + h(ID_c||R_c) \cdot C\right)$ and $1 = \mathrm{verify}\left(ID_s, S_3, \mathrm{n}, (\mathrm{N}, \mathrm{M}), \hat{F}_1, \hat{F}_2, \ldots \hat{F}_s, \hat{\mathrm{T}}, \hat{\ell}_s\right)$. When the equations are true, the data are proven to be complete.

### 5.6 Judgement

When there is a dispute between the client and the cloud server, they each send the latest data information to the judge. The cloud server sends the latest $\mathrm{chal} = \left(\mathrm{i}, s_1, S_2, \mathrm{NI}, \hat{\ell}_c\right)$ and response $\mathrm{re} = \left\{S_3, \mathrm{n}, (\mathrm{N}, \mathrm{M}), \mathrm{wx}', \hat{F}_1, \hat{F}_2, \ldots, \hat{F}_s, \hat{\mathrm{T}}, \hat{\ell}_s\right\}$, $s_3, \ell_c$ to the judge. Then, the judge checks the validity of $\ell_s$. If it is invalid, the cloud server is the winner. Otherwise, the judge checks the validity of $\ell_c$ and re. If one of them is winner, the client is the winner.

## 6 Efficiency Analysis

We perform an efficiency analysis of the storage and computational overhead of the scheme, and compare it with two of the best alternative schemes. Let $T_H, T_{add}, T_{mul}, T_p, T_{exp}$ denote the running time of a hash function instruction, an addition instruction in $G_1$, a multiplication instruction, a bilinear pairing instruction, and an exponentiation instruction. The PRF, PRP and other operations are omitted in our evaluation, because their computational costs are negligible. Suppose the data are split into n blocks. Each block of data is divided into s parts, and the number of challenge blocks is c. Table 1 presents the comparisons between our scheme and two other schemes [1, 7].

From Table 1, we can see that our scheme's computational overhead is the same as scheme [7], so the computational cost is mainly compared with scheme [1]. Our scheme is more efficient because we use a bilinear pairing operation with less computational overhead instead of using exponential operations, and the time expended by $T_H, T_{add}, T_{mul}, T_p$ is less than $T_{exp}$. On the other hand, we can see that when the value of n is fixed, the computational cost of the two schemes is linear with the number of s. Moreover, as the s grows, the growth rate of tag generation computational overhead in scheme [1] is significantly higher than ours. Though comparison, we found that the computational overhead of our scheme is reduced, so our scheme is more efficient. Since both schemes [1, 7] need to maintain the table structure, the scheme, the clients and cloud server do not need to maintain the table structure, and the storage overhead is fixed. Only the number of data blocks n and the seed of the tree can be stored, and the storage overhead is independent of the file size. For this reason, the storage overhead of our scheme is significantly lower than that of the schemes [1, 7], which reduces the storage overhead of the clients and the cloud server.

**Table 1.** Comparison with other schemes.

| Schemes | | Ref. [7] | Ref. [1] | Ours |
|---|---|---|---|---|
| Computational cost in tag generation phase | Client side | $(\mathrm{ns}+2)T_{mul}+nsT_{add}$ $+(n+1)T_{Hash}$ | $nsT_{mul}+nT_{Hash}$ $+\mathrm{n}(\mathrm{s}+1)T_{exp}$ | $(\mathrm{ns}+2)T_{mul}$ $+nsT_{add}+(n+1)T_{Hash}$ |
| Computational cost at generates proof | Cloud side | $(2\mathrm{c}+4)T_{mul}+(c-1)T_{add}$ $+T_{Hash}$ | $(2\mathrm{c}-1)T_{mul}$ $+(c-1)T_{add}+cT_{exp}$ | $(2\mathrm{c}+4)T_{mul}$ $+(c-1)T_{add}+T_{Hash}$ |
| Computational cost at verifying the proof | Client side | $(\mathrm{c}+\mathrm{s}+4)T_{mul}+(c+s+2)T_{add}$ $+3T_{Hash}+T_p$ | $(\mathrm{c}+\mathrm{s}-1)T_{mul}+cT_{Hash}$ $+(c+s+1)T_{exp}$ | $(\mathrm{c}+\mathrm{s}+4)T_{mul}$ $+(c+s+2)T_{add}+3T_{Hash}+T_p$ |
| Storage cost | | ITL list | ORT list | Fixed |

## 7 Summary

The scheme enables the clients and the cloud server to dynamically manipulate outsourced files, making the PDP solution more suitable for practical application. The program supports identity authentication, enabling the PDP solution to eliminate complex certificate management. The program supports non-repudiation and solves the

disputes between the client and the cloud server. the proposed scheme has less computing and storage overhead than other schemes. As such, it has high efficiency.

## References

1. Yan, H., Li, J., Han, J., et al.: A novel efficient remote data possession checking protocol in cloud storage. IEEE Trans. Inf. Forensics Secur. **12**(1), 78–88 (2017)
2. Ateniese, G., Burns, R., Curtmola, R., et al.: Provable data possession at untrusted stores. In: ACM Conference on Computer and Communications Security, pp. 598–609. ACM (2007)
3. Yang, K., Jia, X.: An efficient and secure dynamic auditing protocol for data storage in cloud computing. IEEE Trans. Parallel Distrib. Syst. **24**(9), 1717–1726 (2013)
4. Barsoum, A.F., Hasan, M.A.: Provable multicopy dynamic data possession in cloud computing systems. IEEE Trans. Inf. Forensics Secur. **10**(3), 485–497 (2017)
5. Zhao, J., Xu, C., Li, F., et al.: Identity-based public verification with privacy-preserving for data storage security in cloud computing. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **96**(12), 2709–2716 (2013)
6. Mo, Z., Zhou, Y., Chen, S., et al.: Enabling non-repudiable data possession verification in cloud storage systems. In: IEEE, International Conference on Cloud Computing, pp. 232–239. IEEE, (2014)
7. Wang, F., Xu, L., Wang, H., et al.: Identity-based non-repudiable dynamic provable data possession in cloud storage. Comput. Electr. Eng. **69**, 521–533 (2017)
8. Graham, R., Knuth, D., Patashnik, O.: Specific Math: A Foundation for Computer Science. Posts & telecom press, Beijing (2013)
9. Galindo, D., Garcia, F.D.: A Schnorr-like lightweight identity-based signature scheme. In: Proceedings of Second International Conference on Cryptology in Africa (AFRICACRYPT 2009), 21–25 June, pp. 135–148 (2009)