An access control scheme with fine-grained time constrained attributes based on smart contract and trapdoor

Xuanmei Qin, Yongfeng Huang, Zhen Yang, Xing Li
Beijing National Research Center for Information Science and Technology (BNRist)
Department of Electronic Engineering, Tsinghua University, Beijing 100084, China

***Abstract*—Attribute-based encryption describes the access policy with the attribute information of users. In practice, attributes usually have a certain lifespan. The existing time-based access control methods directly relate attribute keys to time. Thus, under the constraint of fine-grained time, when the attribute expires, the update of key and policy adds a large additional burden to the data user and owner. In this paper, we propose a dynamic attribute-based access control scheme to set a fine-grained valid time period for each attribute, which not only facilitates dynamic data sharing, but also enables flexible attribute revocation. We use smart contract to set valid time period for attributes. It provides smart management on users' attributes and avoids the waiting time of CSP caused by manual operations. We also introduce trapdoor that are indirectly related to time and proxy decryption method to reduce computational cost on data owners and users. Extensive security and performance analysis shows the security strength and effectiveness of the proposed scheme.**

***Index Terms*— Attributed-based encryption, Smart contract, Trapdoor, Proxy decryption**

I. Introduction

AMONG a large number of cloud-oriented data access control mechanisms, attribute-based access control is widely studied for its one-to-many and fine-grained features. The data owner does not need to know the specific identity of the user when encrypting the data, but only needs to use descriptive attributes to formulate the access policy. Data access users can access data associated with access policies according to their attributes. This allows the data owner to decide which attributes a user meets can access his data. However, the existing work only considers the user's entity attributes and ignores the necessary environmental factors. In practice, in many application scenarios, attributes have a certain lifetime. So time is an important attribute constraint. For example, a user subscribes to a video membership for a period of time. During the subscription period, it has a membership attribute. After the subscription period, the membership attribute is revoked and returned to the ordinary members. In this system, the user's attribute of role has a valid time period. The same requirements also exist in applications such as personnel management and copyright document management.

However, the existing time-based mechanisms[1], [2], [3], [4], [5], [6] still have the following problems: 1) Existing solutions are more suitable for coarse-grained time-accurate access validity periods. If the data owner wants to revoke the user from the system at any time, an excessive cost will be required to ensure fine-grained time precision, such as seconds. 2) Time constraints are directly related to attribute policies. Owners upload encrypted data with different strategies at each release time, which makes it impossible for expected users to access data before the corresponding time arrives. Owners repeatedly upload different encrypted versions of the same data, which brings unnecessary cost to data owners. 3) Time constraints are directly related to the attribute keys, which requires the authority to continuously publish the time attribute keys to the user.

This paper aims to implement an attribute-based access control mechanism based on time factor. Only the user satisfying both time and entity attributes can access data correctly. In this paper, an attribute-base access control method based on Smart Contract and Trapdoors(SCT-ABAC) is constructed to solve the above problems in time-based access control. The smart contract[7] is a self-executable code deployed on the blockchain[8], maintained by the miner node, which can effectively reduce the maintenance cost of the attribute by the central authority. And the introduction of trapdoor[9] makes the attributes of access policy not directly related to time, which can effectively reduce the coupling between access policy and time, thus reducing the policy update cost of data owners. Specifically, each outsourced resource is associated with an access policy consisting of a set of attributes, each of which is set a trapdoor. The central authority issues or updates user attributes and their expiration time by calling smart contracts, and issues a trapdoor release key to semi-trusted cloud to release trapdoors. When the user requests data from the cloud, the cloud service invokes the smart contract to query the valid attribute set, and uses the trapdoor decryption key to release these attributes in the access policy.

The main contributions of this paper can be summarized as follows:

1) A fine-grained time-based dynamic attribute access control mechanism, TSC-ABAC, is proposed by introducing trap-

This work is supported in part by the National Key R&D Program of China under Grant 2016YFB0800402, and the National Natural Science Foundation of China under Grant U1536207, U1836204.

The authors are with the Beijing National Research Center for Information Science and Technology (BNRist), Department of Electronic Engineering, Tsinghua University, Beijing 100084, China. Email: qxm17@mails.tsinghua.edu.cn (Xuanmei Qin), yfhuang@tsinghua.edu.cn (Yongfeng Huang)

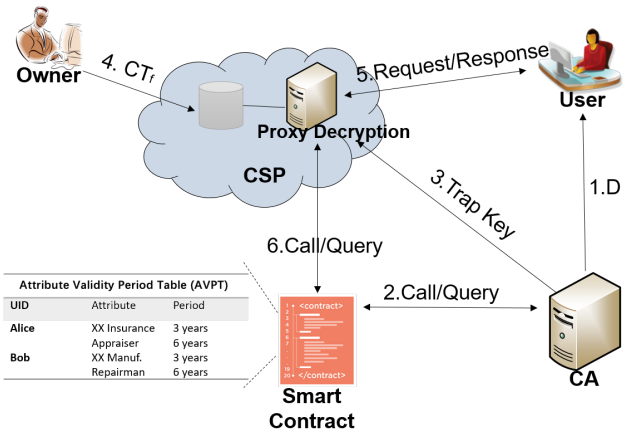

Fig. 1. System model of the proposed TSC-ABAC scheme.

door and smart contract technology in CP-ABE. The validity period of the attribute can be accurate to second.

2) In the encryption process, the introduction of trapdoors that are indirectly related to time reduces the unnecessary cost on the data owner when the attributes expire.

3) We propose a proxy decryption algorithm in the access control scheme. The computing cost of decryption is mainly undertaken by the cloud server. Therefore, the scheme achieves lightweight decryption for users.

The organization of this paper is as follows. In SectionII, we present the system model and state the security assumption. SectionIII gives the detailed algorithm of our TSC-ABAC. SectionIV analyzes the security and performance. Finally, SectionV is a conclusion of this paper.

II. System model

A. System model

As illustrated in Fig. 1. This system model includes the following entities: central authority(CA), data owner(Owner), data user(user), cloud service provider (CSP) and a blockchain network.

Central Authority (CA) is responsible for managing the security of the whole system. It publishes the public parameter PK of the system, issues the attribute private key ASK and trapdoor release key TSK for each user and CSP. The validity period of the user attribute is set in the smart contract. The smart contract plays the role of checking time to determine the valid set of attributes.

The data owner(Owner) uploads data to CSP for sharing. It first constructs access policies according to the attribute set and the trapdoors associated with CSP, and then encrypts data with the access policy and uploads it to CSP.

The cloud service provider (CSP) stores the ciphertexts uploaded by owner, and decides whether to release the trapdoors of corresponding attributes in the access policy according to the smart contract. The result of the proxy decryption is then calculated and sent to user.

The user(User) obtains attribute keys from CA, and requests data from the cloud. Only satisfy the following two conditions can users decrypt successfully: 1) The attribute keys satisfy the data access policy. 2) Current access time is within the validity period of the attributes.

The blockchain network is a decentralized P2P network. Most of the nodes in the network keep the whole chain. The miner node generates blocks through a multi-party consensus mechanism, which ensures the proof-tamper characteristic of blockchain. A smart contract is a piece of logic code deployed on blockchain. It can also be regarded as a special transaction that is sent out and recorded by a miner in a block. The blockchain here mainly refers to the Ethereum blockchain.

In this model, the operations can be divided into two phases: the attribute management phase and the access control execution phase. In the attribute management phase, the CA first issues an attribute key to the user, sets the validity period of the attributes through the smart contract, and issues a trapdoor release key to the CSP. In the access control execution phase, the data owner first uploads the ciphertext, the CSP invokes the contract to obtain the user's valid attribute set, and uses the trapdoor release key and the partial decryption key to perform proxy decryption. Then the user performs final decryption.

A more detailed description of the proxy decryption process is as follows. The ciphertext uploaded by the data owner is generated by an access policy such as "$A3 \wedge (A2 \vee A1)$". Each attribute in the access policy is associated with a trapdoor, which specifies a CSP for proxy decryption. The CSP firstly queries the user's validity period attribute set by invoking smart contract as $A1, A2$, and performs the first step of proxy decryption with the trapdoor release key. This process is equivalent to releasing valid attributes in the access policy. Then the CSP performs the second step of proxy decryption using partial attribute key provided by the user, and computes the intermediate result. The user finally decrypts through the intermediate result.

B. Security assumption

In our proposed method, it is assumed that the CA is fully trusted. The CA is responsible for generating the attribute keys and the trapdoor release keys. It sets the validity period of users' attributes through the smart contract. Like most of attribute-based access control schemes, the CSP is assumed to be semi-trusted, that is, honest-but-curious, which provides reliable storage services and performs computing tasks correctly, but it may attempt to obtain unauthorized data because of personal interests. The user is untrusted. Malicious users attempt to decrypt the ciphertext to obtain unauthorized data. The proposed method can implement a fine-grained time-based access control system. Only users whose attributes are within the validity period and satisfy the access policy can successfully decrypt the ciphertext.

III. Our Construction

A. Mathematical Background

Our proposed method is based on Bilinear Pairing. Let G, G_T be two multiplicative cyclic groups of prime order p, and $e : G \times G \rightarrow G_T$ be a bilinear map, then the following properties are satisfied: (1) Bilinearity:$\forall g, h \in G, a, b \in Z_p^*, e(g^a, h^b) = e(g, h)^{ab}$. (2) Non-degeneracy: $\exists g \in G$, such that $e(g, g)$ is a generator of G_T.

B. Cryptographic Construction

We construct a time-based dynamic attribute access control scheme based on the CP-ABE mechanism. Our scheme includes the following six algorithms:

1) Global Setup. Let G be a multiplicative cyclic group of prime order p. g is a generator of G. $e : G \times G \rightarrow G_T$ is a bilinear map. $H_1 : \{0,1\}^* \rightarrow G^*, H_2 : G_T^* \rightarrow Z_p^*$ are two hash functions. CA randomly selects the parameters α, β, r. Then the public parameters of the system can be defined as

$$GPK = \{G, g, p, h = g^\beta, f = g^{1/\beta}, e(g, g)^\alpha, g^{u_c}\}$$

where u_c is the unique identity of the cloud. u_c and f are used to construct trapdoors. The master key of the system is

$$MSK = \{\beta, g^\alpha\}.$$

2) Key Generation. Let the attribute set of user U_j be S_j. CA randomly selects $u_j \in Z_p^*$ as the unique identity of the user, and chooses the random number $r_i \in Z_p^*$ for each attribute i. Then the private key of user j can be calculated as

$$SK_j = (D, D_i, D_i', \forall i \in S_j),$$

where $D = g^{(\alpha+u_j)/\beta}$, $D_i = g^(u_j)H_1(i)^{r_i}$, $D_i' = g^{r_i}, \forall i \in S_j$. SK_j is sent to U_j through a secure channel between CA and U_j.

3) Trapdoor Key Generation. The CA generates a key $TK = g^{u_c/\beta}$ for releasing the trapdoors in access policy, where u_c is the unique identity of the CSP.

4) Encryption. The data owner first encrypts the data M with the symmetric key $K \in G_T$. For the access policy tree T used to encrypt the key K, each leaf node x is associated with the secret parameters s_x^0 and s_T. s_x^0 is associated with the parent of the node, and s_T is a trapdoor parameter. When x is the root node R of the tree, parameter $s \in Z_p^*$ is randomly selected such that $s_R^0 = s$ is the primary key of the data. Starting from the root node, assign secret parameters to each node. Then randomly select a number $d \in Z_p^*$ to calculate the trapdoor as

$$TS = (A = g^d, B = s_T + H_2(e(g^{u_c}, f)^d)),$$

where u_c is the CSP specified by the data owner to conduct proxy decryption. Let Att_x be the corresponding attribute of leaf node x. The final ciphertext uploaded is computed as

$$CT = (T, C', \overline{C}, C, TS, C_x, C_x', \forall x \in T).$$

among them, $C' = Enc(M, K), \overline{C} = K \cdot e(g, g)^{\alpha s}, C = h^s, C_x = g^{s_x^0 \cdot s_T}, C_x' = H_1(Att_x)^{s_x^0 \cdot s_T}$.

5) Proxy Decryption. The CSP obtains partial decryption key D_i, D_i' from the user, and performs a bottom-up calculation process. For the leaf node x, if $i \in S_j$

$$F_x = \frac{e(C_x, D_i)}{e(C_x', D_i')} = \frac{e(g^{s_x^0 \cdot s_T}, g^{u_j} H_1(i)^{r_i})}{e(H_1(Attr_x)^{s_x^0 \cdot s_T}, g^{r_i})} = e(g, g)^{u_j s_x^0 \cdot s_T}.$$

Otherwise, let $F_x = \perp$. Then, the CSP queries the attribute set of the user that has not expired through a smart contract. Smart contracts are deployed on blockchain, which have the characteristics of non-tampering, thus ensuring the reliability of the results of query. If x belongs to the attribute set, the trapdoor is released as $TS' = B - H_2(e(TK, A)) = s_T$. Otherwise $TS' = 1$. CSP uses TS' to recalculate F_x, and obtains

$$\tilde{F}_x = {F_x}^{TS'} = \begin{cases} e(g, g)^{u_j s_x^0} & x \text{ is in validity period} \\ e(g, g)^{u_j s_x^0 \cdot s_T} & x \text{ is not valid} \end{cases}$$

If the user's attribute satisfies the access policy and the corresponding attribute is within the validity period, the CSP can correctly perform the proxy decryption process. For the root node R, if $F_R \neq \perp$, then $F_R = e(g, g)^{u_j s}$. Even if the attacker obtains F_R, but there is no decryption key D, the plaintext still cannot be recovered. Finally, the CSP sends $CT' = (C', \overline{C}, C, F_R)$ to the user.

6) Decryption. The user recovery message content by calculating

$$K' = \frac{\overline{C}}{(e(C, D)/F_R)} = K, M' = Dec(C', K) = M.$$

If the user's attributes do not satisfy the access policy or the validity period, the decryption fails.

C. Smart Contract Construction

Compile a smart contract used to manage the validity period of users' attributes and deploy it to the blockchain. Once the smart contract is deployed successfully, the miner will permanently record it in the blockchain and return the contract address to the contract creator.

1) Attribute Validity Period Table (AVPT). In order to manage the lifetime of attributes, we introduce an attribute period table, which defines the validity period of each attribute of the user. The time unit can be set as "second, minute, hour, day, week, year". The implementation of the attribute time table uses the mapping type, which has a unique storage model in the blockchain. The CA assigns each user a set of attributes and the lifetime of these attributes. To illustrate, consider the following application scenarios. As shown in Fig. 2 , Alice is a used car appraiser of an insurance company. Bob is a car repairer of an automobile manufacturer. Each attribute of the user is valid for a predetermined period of time.

2) Contract Deployment. As mentioned above, the smart contract is mainly used to manage the APT. The corresponding smart contract is briefly described in the algorithm1. The CA sets, updates, and deletes terms in AVPTs through the functions of addAtt, updateAtt, and deleteAtt respectively. The CSP and CA query the unexpired attribute set through the

Attribute Validity Period Table (AVPT)		
UID	Attribute	Period
Alice	XX Insurance	3 years
	Appraiser	6 years
Bob	XX Manuf.	3 years
	Repairman	6 years

Fig. 2. An example of AVPT.

checkAtt function. As a trusted authority, the CA's address and deployed contract address are published to the public. To protect users' privacy, the CA can specify CSPs that perform checkAtt function.

IV. Performance Analysis

This section implements our proposed algorithm and gives an intuitive performance evaluation. The experiment consists of two parts, cryptographic algorithms and smart contract deployment. Performance assessment will quantify the time cost of cryptographic algorithms and smart contract operations, respectively. We simulate the cryptographic algorithm based on python-charm library [1] and compare it with the CP-ABE algorithm[3] without trapdoors to evaluate its time cost. Smart contracts are compiled and deployed through the Remix[2] platform. Remix is an open source development environment for solidity-based smart contract. It provides basic functions such as compiling and deploying contracts to a local or test Etherum network, and executing contracts. We interact with a smart contract via the Web3.js API.

Performance of Cryptographic algorithms. In evaluating the performance of the proposed attribute encryption mechanism, we use the most complex access strategy "$att_1 \quad AND \quad att_2 \quad AND ... att_N$". When generating a ciphertext, each attribute is set a trapdoor related to CSP. The number of trapdoors and the number of attributes in the policy are both N, and the trapdoor can only be released by the specified CSP. TableI shows the time performance when the number of attributes is 20 and each algorithm runs 100 times. In the proposed method, a trapdoor is a common parameter for all users. It specifies the CSP that can release the trapdoor and perform proxy decryption. Therefore, the CA only needs to calculate and issue a trapdoor release key to the CSP, and does not need to issue the time-related key periodically. As shown in Fig.3, compared to the classic CP-ABE without trapdoors, with the number of trapdoors increases, the additional encryption burden on the data owner is negligible. During the decryption process, the CSP releases the trapdoor and performs proxy

TABLE I
Time cost of Cryptographic algorithms(in seconds)

Algorithm	Setup	keyGen	Encrypt	Decrypt
Max Time	0.0228	0.0228	0.1663	0.0020
Min Time	0.0139	0.0139	0.0755	0.0006
Average Time	0.0187	0.0187	0.1042	0.0014

[1]https://github.com/JHUISI/charm

[2]http://remix.ethereum.org

Algorithm 1 Smart Contract on AVPT

Require: Function name, invoke parameters
Ensure: Setting up functions

```
Structure User {
        string[] Attr;
        uint[] period;
        uint[] start;
}
function APT()
    CA_address ← sender.address
    mapping(string ⇒ User) users
    address CSPs[]
end function
function ADDATT(Uid, Attr, period)
    if message.sender≠CA_address then return 0
    else
        users[Uid].Attr.push(Attr)
        users[Uid].period.push(period)
        users[Uid].start.push(now)
    end if
end function
function UPDATEATT(Uid, oldAtt, newAtt, newPeriod)
    if message.sender≠CA_address then return 0
    else
        if oldAtt=users[Uid].Attr[i] then index=i
        end if
        users[Uid].Attr[index] ← newAtt;
        users[Uid].period[index] ← newPeriod;
    end if
end function
function DELETEATT(Uid, Att)
    if message.sender≠ CA_address then return 0
    else
        if oldAtt=users[Uid].Attr[i] then index=i
        end if
        users[name].strArr[i]=users[name].Attr[i+1]
        users[name].period[i]=users[name].period[i+1]
        users[name].start[i]=users[name].start[i+1]
        len–
    end if
end function
function SETCSPs(CSP_address)
    if  message.sender≠ CA_address then return 0
    else
        CSPs.push(CSP_address)
    end if
end function
function CHECKATT(Uid)
    if message.sender∉ CSPs then return 0
    else
        uint[] index
        if current time is not expired then index.push(j)
        end if
        deleteAtt(Uid, users[Uid].Attr[index])
        return (users[Uid].Attr,users[Uid].period,
                users[Uid].start)
    end if
end function
```

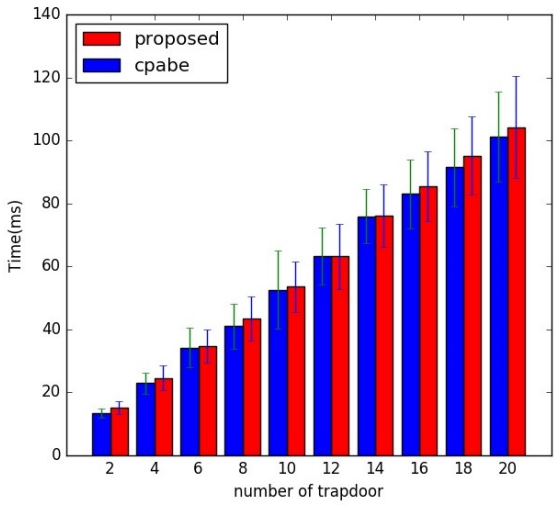

Fig. 3. Time cost of Encryption process versus number of trapdoors

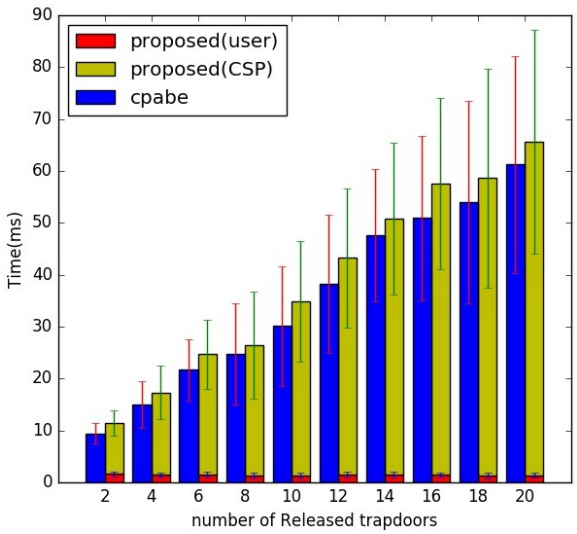

Fig. 4. Time costs of proxy Decryption and Decryption process versus number of trapdoors

decryption for each file. Here, the experimental environment of proxy decryption is the same as that of the other cryptographic algorithms. The performance is shown in Fig.4. With the number of traps released increases, CSP bears the main computational burden, and users only need to pay less computational effort. Therefore, the proposed method provides a flexible and lightweight access control system.

Performance of Smart contract . In order to improve the efficiency of contract's development, we choose to build a local Ethereum test environment with Ganache which is officially supported by Ethereum. Web3.js is the official Javascript API of Ethereum. It can be used to interact with Ethereum Smart Contracts. Since Solidity only provides accuracy of time to the second, we use javascript to get the time cost for operations related to smart contract. TableII summarizes the time cost of each operation in smart contract. Each call to function is equivalent to submit a transaction, and the result of the call is returned after the transaction is successful. The

TABLE II
TIME COST OF SMART CONTRACT

Algorithms	Time Cost(in seconds)
addAtt	0.3605
checkAtt	0.3017
updateAtt	0.3204
deleteAtt	0.3269

time cost of all operations is in seconds and the measurement is the average of 100 times of function calls.

V. CONCLUSION

In this paper, an attribute-based access control scheme based on time domain, SCT-ABAC, is proposed under the collaboration of cloud and blockchain. This solution introduces proxy decryption in the access control system, and the decrypted computing task is mainly undertaken by the cloud server. The analysis shows that the user's computation cost in the decryption process is almost a constant. Therefore, the scheme achieves lightweight decryption and can be applied to situations where data requesters have limited resources, such as IoV and IoT devices. The central authority specifies the attributes of the user and its expiration date through smart contract. The operation functions of AVPT defined in smart contract is executed by miner nodes in blockchain. The miner nodes need to bear the computational and storage cost of the AVPT. The distributed ledger and decentralized consensus of the blockchain ensure that the AVPT is secure, reliable and proof-tamper. Our scheme can realize that only the users' attributes within the validity period satisfy the access policy can decrypt correctly, thus ensuring data confidentiality.

REFERENCES

[1] E. Androulaki, C. Soriente, L. Malisa, and S. Capkun, "Enforcing location and time-based access control on cloud-stored data," in *2014 IEEE 34th International Conference on Distributed Computing Systems*, June 2014, pp. 637–648.

[2] M. Pirretti, P. Traynor, P. McDaniel, and B. Waters, "Secure attribute-based systems," in *Proceedings of the 13th ACM Conference on Computer and Communications Security*, ser. CCS '06. New York, NY, USA: ACM, 2006, pp. 99–112.

[3] J. Bethencourt, A. Sahai, and B. Waters, "Ciphertext-policy attribute-based encryption," in *2007 IEEE Symposium on Security and Privacy (SP '07)*, May 2007, pp. 321–334.

[4] K. Xue, J. Hong, Y. Xue, D. S. Wei, N. Yu, and P. Hong, "Cabe: A new comparable attribute-based encryption construction with 0-encoding and 1-encoding," *IEEE Transactions on Computers*, vol. 66, no. 9, pp. 1491–1503, Sept. 2017.

[5] Q. Liu, G. Wang, and J. Wu, "Time-based proxy re-encryption scheme for secure data sharing in a cloud environment," *Information Sciences*, vol. 258, pp. 355 – 370, 2014.

[6] Q. Liu, C. C. Tan, J. Wu, and G. Wang, "Reliable re-encryption in unreliable clouds," in *2011 IEEE Global Telecommunications Conference - GLOBECOM 2011*, Dec 2011, pp. 1–5.

[7] N. Szabo, "Secure property titles with owner authority," https://web.archive.org/web/20140115142013/http://szabo.best.vwh.net/securetitle.html, accessed 2019.

[8] S. Nakamoto, "Bitcoin: a peer-to-peer electronic cash system," https://bitcoin.org/en/bitcoin-paper, accessed 2019.

[9] G. D. Crescenzo, R. Ostrovsky, and S. Rajagopalan, "Conditional oblivious transfer and timed-release encryption," in *Advances in Cryptology - EUROCRYPT '99, International Conference on the Theory and Application of Cryptographic Techniques*, ser. Lecture Notes in Computer Science, vol. 1592. Springer, 1999, pp. 74–89.