

Fair Identification

Omkant Pandey¹, Julien Cathalo^{2*}, and Jean-Jacques Quisquater²

¹ Department of Computer Science, UCLA
omkant@cs.ucla.edu

² UCL Crypto Group, Belgium
{cathalo, quisquater}@dice.ucl.ac.be

Abstract. This paper studies a new problem called fair identification: given two parties, how should they identify each other in a fair manner. More precisely, if both parties are honest then they learn each other's identity, and if anyone is cheating then either both of them learn each other's identity or no one learns no information about the identity of the other. We propose a security model and a provably secure optimistic fair identification protocol.

1 Introduction

Suppose that Alice and Bob are interested in knowing each other but each of them is hesitating in revealing his/her identity first. How should they identify each other so that fairness is guaranteed for both parties i.e. if both of them are honest, they learn each other's identity; if any one of them cheats then either both of them learn each other's identity or no one learns no information about the identity of the other. This problem is termed as *fair identification*.

We will be interested in a protocol which ensures the following:

- If A and B are honest, both of them learn each other's identity.
- If anyone is cheating, either no one learns anything about the identity of the other or both of them learn each other's identity.
- Identities of A and B remain secret to an outsider against active attacks.
- A third party is needed only in case of disputes i.e. when cheating occurs (in other words, the protocol should be optimistic).

Harder variants of this problem are also possible: for example, one could consider concurrent attacks instead of active attacks, but in this paper we will focus on active attacks only.

Several problems similar to fair identification have been studied in the literature. It is thus a natural approach to examine a few kinds of cryptographic primitives used to solve such problems to see if they can trivially achieve fair identification.

* supported by *Walloon Region / WIST-MAIS project*

- In a mutual authentication scheme [10, 6, 23], two parties authenticate each other, but fairness is not ensured.
- In a fair exchange of signatures protocol [1, 20, 16, 2], each party obtains the other’s signature in a fair manner. There is a fundamental difference between exchanging a signature and exchanging one’s identity. Fair exchange of signatures is based upon the concept of verifiability of signatures without actually completely revealing them [13, 3, 20]. But a successful verification always confirms the identity of the other party. Thus, fair exchange of digital signatures does not seem to provide any trivial solution to fair identification.
- Identity escrow schemes [25] allow an entity A to send some information to B that commits to A ’s identity, meaning that this information would allow an authorized third party to recover A ’s identity. A and B could run a fair identification protocol as follows:
 1. A runs the identity escrow protocol with B .
 2. B confirms his true identity to A .
 3. A confirms her true identity to B in similar way.

If A is cheating, B can go to the escrow agent with transcripts of identity escrow protocol in step 1 to obtain A ’s identity. In this protocol, eavesdroppers can learn the identities of A and B . Simply encrypting the communication does not thwart active attacks. Thus this protocol does not satisfy the requirements for fair identification.

Since there seems to be no trivial way to achieve fair identification, we propose a new scheme. Rather than build a scheme from scratch, we use existing cryptographic primitives and combine them to design a fair identification scheme.

This paper is organized as follows. Section 2 lists the cryptographic primitives that we use as building blocks for our protocol. Section 3 defines a security model for fair identification. Section 4 describes our fair identification protocol. Section 5 discusses a few variants of the initial problem.

2 Building Blocks

In this section, we introduce the cryptographic primitives that we use as building blocks for our fair identification protocol. These building blocks are a signature scheme with a special feature, a public key encryption scheme, and a group signature scheme. This section is not necessary for understanding the security model of section 3 and hence can be skipped. However, it is necessary for the protocol (section 4).

2.1 Signatures with Key-Independent Coupons

Some signature schemes have the interesting feature that a part of the signature can be computed prior to the knowledge of the message to sign. They are sometimes called on-line/off-line signatures [28] or coupon-based signatures, and the pre-computed part is called the coupon.

In this paper, we additionally require that the coupon can be computed without knowing the signing key. We call such a scheme a signature scheme with key-independent coupons.

Many known signature schemes satisfy this requirement; in fact, any signature scheme obtained by applying the Fiat-Shamir heuristic [21] does (the coupon is the commitment of the corresponding identification scheme), like the Schnorr signature scheme [27] for example.

More formally, a signature scheme with key-independent coupons is a tuple of algorithms $\mathcal{SS} = (\mathcal{K}_S, \mathcal{S}, \mathcal{V})$ satisfying the usual properties of a signature scheme where: \mathcal{K}_S , \mathcal{S} , and \mathcal{V} are key-generation, signing and verification algorithms respectively. Additionally, \mathcal{S} internally works as follows ($s =$ signing key and $\hat{st} =$ some state information, $m =$ message to be signed): Algorithm $\mathcal{S}(m, s) : \{(x, \hat{st}) \leftarrow \mathcal{S}_x(); y \leftarrow \mathcal{S}_y(x, \hat{st}, m, s); \text{Return}(x, y); \}$.

Here, \mathcal{S}_x is the algorithm that generates the coupon and \mathcal{S}_y is the algorithm that generates the remaining part of the signature.

NOTATION: Let $cert$ denote the certificate for the public key of a signature scheme with key-independent coupons. We assume two algorithms Extract and Valid such that: $\text{Extract}(cert)$ extracts the public key from $cert$ and $\text{Valid}(cert)$ verifies if $cert$ is valid (output 1) or not (output 0). Furthermore, we assume that all certificates are of the same size and that all signatures are also equal in size (if not, add leading 0s to make them equal to the maximum possible size).

2.2 Public Key Encryption in Multi-User Setting

By \mathcal{PE} we denote an IND-CCA secure public-key encryption scheme secure in a multi-user setting (Bellare et al [5]).

Recall that a public-key encryption scheme $\mathcal{PE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ consists of three algorithms. The *key generation* algorithm \mathcal{K} is a randomized algorithm that takes nothing as input and returns a pair (pk, sk) of matching public and secret keys; we write $(pk, sk) \stackrel{R}{\leftarrow} \mathcal{K}()$. The *encryption* algorithm \mathcal{E} is a randomized algorithm that takes the public key pk and a *plaintext* M (from the message space $\text{MsgSp}(pk)$) to return a *ciphertext* C' ; we write $C' \stackrel{R}{\leftarrow} \mathcal{E}_{pk}(M)$. The *decryption* algorithm \mathcal{D} is a deterministic algorithm that takes the secret key sk and a ciphertext C' to return the corresponding plaintext M (or a special symbol \perp if C' is invalid); we write $M \leftarrow \mathcal{D}_{sk}(C')$.

Two ideal examples of \mathcal{PE} are: Cramer-Shoup [19, 5] and RSA-OAEP [9, 7, 29, 22].

2.3 Group Signatures as Verifiable Commitments to Identity

Group signatures allow a group member to sign anonymously on behalf of the group. If needed, the signature can be opened by a trusted third party, called group manager, to reveal the identity of the signer. For an in-depth discussion on group signatures see [14, 12, 4, 8, 11].

Informally, a group signature scheme $\mathcal{G} = (\text{Setup}, \text{Join}, \text{Sign}, \text{Verify}, \text{Open})$ is a 5-tuple of algorithms, where:

–**Setup** is the algorithm which takes no input. It initializes the system and outputs the group public key GPK , secret data for the group manager, and any other parameters needed.

–**Join** is an interactive protocol executed between the group manager and a user (say A). As a result, A learns his secret data g_A for generating group signatures and the group manager might also learn some data to aid him later in opening the signatures if required.

–**Sign** is the signing algorithm. It takes as input the message m to be signed and the secret data g_A of any user A to produce a group signature σ_A .

–**Verify** is the algorithm to verify the correctness of a group signature on a given message. It takes as input the message m , the signature σ and the group public key GPK ; it outputs 1 if σ is a valid group signature on m , and 0 otherwise.

–**Open** is the algorithm that only the group manager can use to identify the signer of a particular group signature. It takes as input the signature σ , manager's secret data for opening group signatures and perhaps some other information; its output is a proof identifying the signer of the signature.

For notation, $\sigma_A(m)$ will denote the group signature of A on message m . When only σ_A is written, it means that it is a group signature of A on whatever message and that opening it would identify A as its signer.

The following properties are desirable for group signatures: *Correctness* - group signatures produced using **Sign** are always accepted by **Verify**. *Unforgeability* - only group members can sign efficiently; *Anonymity* - given a group signature, it is computationally hard to identify its signer for everyone but the group manager; *Unlinkability* - deciding whether two valid group signatures were computed by the same group member, is computationally hard; *Exculpability* - Neither a group member nor the group manager can produce a group signature on behalf of any other member. *Openability* - The group manager is always able to open and identify the actual signer of a valid group signature; and *Coalition-Resistance* - A colluding subset of group members (even if comprised of the entire group) cannot generate a group signature that the group manager cannot open.

Let us explain why and how we use group signatures in our scheme. Our approach to design a fair identification scheme is inspired by the way fair exchange of signature schemes are built. In order to fairly exchange signatures, users need a way to commit to their signatures such that a third party can reveal the signature if needed (this was formalized by Dodis and Reyzin [20] and called a verifiably committed signature scheme). Similarly, in order to fairly exchange identities, users need a way to commit to their identities. Group signatures solve this problem; they can be seen as "verifiable commitments to identity", or VCI for short.

3 A Security Model for Fair Identification

In this section we introduce the security notions that we require for a fair identification scheme.

3.1 Setting

Parties A and B are willing to fairly identify each other. The trusted third party is T . In order to simplify the description of the protocol, T will have two functions: certificate generation and dispute resolution. We restrict ourselves to the case of active attacks meaning that each player communicates with only one player at a time.

We need to consider coalition attacks, where the adversary is allowed to form coalitions with any number of users. We shall treat each coalition of adversaries as a *single* adversary who will be considered as identified if any one in the coalition is identified with overwhelming probability. This approach of identifying at least one adversary has been widely used in traitor tracing [15, 17, 18, 24] and also in group signatures (see the coalition resistance property).

Because now each coalition can be replaced by a single adversary, we can assume that *users form no coalitions* at all.

3.2 Canonical Protocol

We now present a canonical protocol for identification. For the sake of simplicity, we omit exchanges that happen before B commits to his identity. The sketch of this protocol is the following: in the first step, B commits to his identity. In the second step, A reveals and proves her identity. In the third step, B reveals and proves his identity.

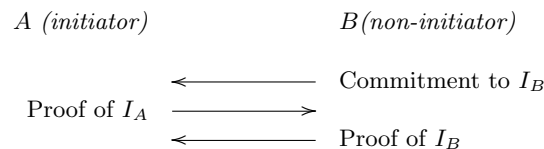


Fig. 1. Canonical fair identification protocol

3.3 Definitions

In a real world scenario, the adversary might interact with any of the users at will, eavesdrop, modify/stop the data, etc. To simulate all these actions, we use oracles.

An *identity oracle* \mathcal{O}_i for identity I_i essentially simulates the behavior of a honest user whose identity is I_i and hence knows the required secret s_i to prove its identity and any other secret data necessary for generating its own VCI. Each oracle is capable of executing a fair identification protocol with any other oracle or user.

Definition 1 (Identity Oracles) *An identity oracle \mathcal{O}_i , is the simulation of a honest user with identity I_i equipped with all necessary secrets required to execute the fair identification protocol. Besides the messages of the fair identification protocol, the oracle understands the following instructions – here, \mathbb{O} is either an identity oracle $\mathcal{O}_j \neq \mathcal{O}_i$ or the player who issues these instructions:*

- *START(\mathbb{O}): when this instruction is issued to \mathcal{O}_i , it starts executing a fair identification protocol with \mathbb{O} .*
- *TRANSCRIPT(\mathbb{O}): when this instruction is issued to \mathcal{O}_i , it provides the issuer with the transcripts of a fair identification protocol run between \mathcal{O}_i and \mathbb{O} .*

In both cases, \mathcal{O}_i will be the initiator. During any live session, the oracle either sends an appropriate message or waits for an appropriate message. It stops whenever an invalid message arrives or if the protocol completes successfully.

The START instruction allows simulating live sessions whereas TRANSCRIPT instruction simulates access to the old transcripts. The time for START operation will be one unit. When an adversary eavesdrops and gathers transcripts, gathering one transcript in real world lasts as long as one fair identification run between two parties. Once the adversary has all the transcripts it needs, it can access them in constant time. But for this, he must still gather and store transcripts, and the time taken for that should actually be counted. Thus, the time for obtaining an answer for a TRANSCRIPT instruction is essentially the run-time for one protocol run.

By q_i , denote the total number of all those instructions (START,TRANSCRIPT), in which the identity oracle \mathcal{O}_i appears (either initiator or non-initiator). Let q_s denote the maximum value of q_i over all i , i.e. $q_i \leq q_s, \forall i, 0 \leq i \leq u-1$. We will expect the protocol to be secure for large values of q_s .

Now we are ready to present a game for the adversary and formally define the notion of fairness. For the rest of the paper, C denotes the adversary and his identity is I_C . By ε we denote an empty string. We assume that there are $u+1$ users in the system including the adversary. Excluding C , there are u users with identities denoted by I_0, I_1, \dots, I_{u-1} . To simulate them, we assume u identity oracles: $\mathcal{O}_0, \mathcal{O}_1, \dots, \mathcal{O}_{u-1}$. By S we represent the set $\{\mathcal{O}_0, \mathcal{O}_1, \dots, \mathcal{O}_{u-1}\}$ and access to S means access to each of its element oracles.

Definition 2 (Fairness-game) *Let $\mathcal{O}_0, \mathcal{O}_1, \dots, \mathcal{O}_{u-1}$ be the identity oracles corresponding to the identities I_0, I_1, \dots, I_{u-1} and $S = \{\mathcal{O}_0, \mathcal{O}_1, \dots, \mathcal{O}_{u-1}\}$. Adversary C is asked to choose any two identities, say I_0 and I_1 of his choice from the set $\{I_0, I_1, \dots, I_{u-1}\}$. Now one of these two identities is selected at random, and represented by I_b where $b \stackrel{R}{\leftarrow} \{0, 1\}$. An identity oracle Ch with identity I_b*

is generated as a challenge oracle for the adversary. The adversary is given access to Ch and S . Adversary knows the identities of all the oracles in S but not of Ch . The adversary can instruct these oracles and ask for transcripts of any communication and/or for opening new sessions with anyone. It is mandatory that each oracle executes only one session at a time, and that it can be involved in no more than q_s instructions during the entire game.

Oracle Ch differs from other identity oracles in one manner: it keeps a state bit ID , initialized to 0 at the start of the game, which changes automatically along the game progress, as follows:

- Initiator case: when Ch is initiator in a session, a fresh session-specific state $St = (\tilde{a}, \tilde{b})$ is created at the start of the session, where \tilde{a} is a bit-string with initial value ε and \tilde{b} is a single bit with initial value 0. If V is the VCI received by Ch in the first step and V is valid, then \tilde{a} is set to V . Value of \tilde{b} becomes 1 if an invalid message arrives in the third step or if this step never occurs. If the third step succeeds, then let I' be the identity whose proof was accepted by Ch in this step. Then,

$$ID = 1 \text{ if } \begin{cases} I' = I_C \\ \text{or} \\ St = (VCI_C, 1) \text{ i.e. } \tilde{a} = VCI_C \text{ and } \tilde{b} = 1 \end{cases}$$

- Non-initiator case: if Ch is the non-initiator, consider the second step of canonical FIP. This step succeeds if the initiator provides appropriate data. If the initiator does not provide appropriate data, then at a later point in time, oracle might receive this data from T during the dispute resolution when initiator approaches T . Be it through any of these cases, let I' represent the identity whose proof is accepted by the oracle. Then,

$$ID = 1 \text{ if } I' = I_C$$

Adversary wins the game if at the end of the game it outputs a bit b' such that: $(b' = b) \wedge (ID = 0)$.

Definition 3 (Fairness) Let \mathcal{W} denote the event that adversary C wins the fairness-game. An FIP is said to ensure fairness if any polynomial time adversary has only negligible advantage in fairness-game, where the advantage of the adversary is defined as,

$$\mathbf{Adv}_C^{Ch,S} = 2 \cdot \Pr[\mathcal{W}] - 1 = 2 \cdot \Pr[(b' = b) \wedge (ID = 0)] - 1$$

Informally, the idea behind the fairness-game is that adversary is allowed to pick up any two oracles he would like to attack. One of these oracles is picked at random as challenge oracle for the adversary and the adversary is asked to guess the identity of the challenge oracle with probability acceptably larger than $\frac{1}{2}$.

In this process, the adversary is not allowed to give away his own identity. This is formalized by ID . In the initiator case, $\tilde{b} = 1$ means that the oracle detects cheating and hence will approach the third party. Adversary will be identified

only if V is his own vci. Hence, $St = (\text{vci}_C, 1)$ means that C is identified through the third party. In both cases, $I' = I_C$ simply means that adversary himself gave out his identity. Thus ID is just like a flag which when set, represents that C 's identity is disclosed to the challenge oracle.

4 A Fair Identification Protocol

In this section, we describe a fair identification protocol. We start by giving a sketch of our protocol. First, A generates and sends the coupon of his signature. Now, B , in the second step, generates a group signature on A 's coupon and sends it to A . A now has the vci of B and hence it now sends his identity and the remaining part of the signature. B verifies the signature and then sends his identity and signature to A . To remain anonymous to outsiders, they encrypt the communication using temporary keys (using \mathcal{PE}). To avoid active attacks, these keys are signed under the group signature of B .

Whenever not mentioned, the security parameter and other system parameters should be assumed implicitly available. INITIAL-SETUP is needed for each user to learn his corresponding secrets and to designate the trusted third party. More users can join at any time. EXCHANGE is the main protocol for exchanging identities. RESOLVE is the last component of our FIP, required only in case of dispute. Let T denote the trusted third party which will also take the role of group manager in the group signature scheme.

INITIAL-SETUP:

System specific parameters:

1. T declares: (a) A secure signature scheme with key-independent coupons $\mathcal{SS} = (\mathcal{K}_S, \mathcal{S}, \mathcal{V})$, and (b) An IND-CCA secure public-key encryption scheme $\mathcal{PE} = (\mathcal{K}, \mathcal{E}, \mathcal{D})$ to be used by each player, whenever needed.
2. Decide a secure group signature scheme to be used, $\mathcal{G} = (\text{Setup}, \text{Join}, \text{Sign}, \text{Verify}, \text{Open})$ with T being the group manager. T creates an instance of \mathcal{G} by running the procedure **Setup**. T learns the secrets corresponding to the group manager and let GPK be the group public key.

User specific parameters:

1. Each user U first decides his public and private keys, I_U and s_U respectively, for the signature scheme \mathcal{SS} by running \mathcal{K}_S . U proves to T that I_U is his public key.
2. Now U runs the **Join** protocol of \mathcal{G} , with T , to learn his group specific secret g_U needed to produce the group signatures.
3. T generates a certificate $cert_U$ mentioning that person with public key I_U is registered with T .

EXCHANGE:

Parties A (initiator) and B (non-initiator) execute the following steps to identify each other:

1. A : $(x_A, \hat{st}) \leftarrow \mathcal{S}_x()$, $(pk_A, sk_A) \leftarrow \mathcal{K}()$. A sends x_A, pk_A to B .
2. B : $(pk_B, sk_B) \leftarrow \mathcal{K}()$, $\sigma_B \leftarrow \text{Sign}(x_A \| pk_A \| pk_B, g_B)$. Send pk_B, σ_B to A .
3. A : If $\text{Verify}(x_A \| pk_A \| pk_B, \sigma_B, GPK) = 1$ then
 $y_A \leftarrow \mathcal{S}_y(x_A, \hat{st}, \sigma_B, s_A)$, $\psi_A \leftarrow \mathcal{E}_{pk_B}(y_A, cert_A)$, Send ψ_A to B .
else STOP.
4. B : $(y_A, cert_A) \leftarrow \mathcal{D}_{sk_B}(\psi_A)$, $XY_A \leftarrow (x_A, y_A)$, Verify $cert_A$ and if valid, extract I_A .
If $\mathcal{V}(\sigma_B, XY_A, I_A) = 1$ then
 $XY_B \leftarrow \mathcal{S}(\sigma_B, s_B)$, $\psi_B \leftarrow \mathcal{E}_{pk_A}(XY_B, cert_B)$, Send ψ_B to A .
else STOP.

Final test, A : $(XY_B, cert_B) \leftarrow \mathcal{D}_{sk_A}(\psi_B)$, Verify $cert_B$ and if valid, extract I_B and check that $\mathcal{V}(\sigma_B, XY_B, I_B) = 1$. If tests do not succeed, A goes to T to execute RESOLVE.

RESOLVE:

Party A claiming to be cheated, presents σ_B to T and the corresponding message components x_A, pk_A, pk_B , full signature XY_A on σ_B and identifies herself as A to T . T sends a proof identifying the signer (B) of σ_B to A and sends the full signature XY_A to B .

For a security proof of this protocol, please see the full version of this paper [26].

5 Extensions and Future Work

In this section, we discuss a few variants to the problem of fair identification. While not all of these variations might find a practical application, they constitute an interesting challenge.

5.1 Transferable Proofs of Identity

One could imagine a scenario where users want to fairly exchange *transferable* proofs of identity. In this case, the precise statement of the problem will be: how should two parties identify each other so that either each party learns a transferable proof of the other's identity or none of them learns nothing about the identity of the other.

In our fair identification protocol, the proofs of identity that users get if the protocol ends normally are indeed transferable (because those proofs are signatures), but it does not necessarily mean that our protocol is a fair exchange of identity proofs. This property can be ensured if group signatures can be opened

in a transferable manner, i.e. when the third party opens the group signature, it outputs a transferable proof that identifies its signer. Though it might not be possible for every group signature scheme, it is indeed the case in modern schemes. For example, in ACJT scheme [4], this proof is actually an interactive zero-knowledge proof of equality of two discrete logarithms, between the group manager and the party interested in identifying the signer. This can be made transferable by applying the Fiat-Shamir heuristic. Thus, if cheating occurs in our protocol, the cheated party can also get the transferable proof which will be nothing but the group signature together with a transferable proof identifying its signer (obtained from T).

Now let us consider a stronger requirement: the proofs obtained with the help of the third party should be at least computationally indistinguishable from the proofs obtained directly from the concerned player. This requirement has an interesting link with the fair exchange of signatures where signatures opened by the third party should be indistinguishable from signatures computed by the signer.

Such indistinguishability cannot be achieved using our protocol. It would be interesting to see whether existing protocols for fair exchange of signatures can be modified to provide fair identification too.

5.2 Nontransferable Proofs of Identity

If the fair identification protocol between A and B ended successfully, A revealed her identity to B , so B can claim to another user that he performed an exchange with A . But A might want to be sure that B cannot *prove* that they indeed met. A protocol that would ensure this for A and B would be a fair exchange of nontransferable proofs of identities.

More precisely, if A and B execute the fair identification protocol and identify each other, then after the protocol completes, transcripts of the run prove nothing to anyone – transcripts could have very well been simulated by A (or B) himself.

Our protocol cannot be used to guarantee nontransferable proofs of identity. One might think of using interactive proof protocols instead of signatures to achieve this goal. Although this could work, neither this construction nor its security proof are trivial.

5.3 Conditional Fair Identification

Assume a situation where the parties know in advance whom they want to identify fairly. More precisely, A is willing to achieve fair identification only with B , and B is willing to achieve fair identification only with A . With very little tweaking, our protocol should work for this kind of problem. However a more general situation is the following one: A is willing to achieve fair identification with B if and only if B satisfies *some condition*, say \mathcal{C}_B . Similarly, B is willing to do fair identification with A if and only if A satisfies some condition, say \mathcal{C}_A .

We term it as conditional fair identification. Our protocol may work for this, depending upon what these conditions are.

5.4 Perfect Fairness

In this paper, the coalition of adversaries was considered to be one single adversary. Following an approach used in traitor tracing schemes, we aimed at identifying at least one user in the coalition. However, a stronger notion of fairness is possible where one could aim at identifying *each* player in the coalition. Precisely, perfect fairness for A would mean that if A is identified by any player B in a particular run of the protocol, A will also identify *exactly* B . With this perfect fairness notion, it should not matter whether B is in a coalition or not. In order to achieve perfect fairness, it is necessary to use non-transferable proofs; otherwise, just one person could interact with A and obtain a transferable proof of identifying A and then show it to everyone in the coalition. Perfect fairness seems to be the most challenging property to ensure.

Acknowledgements

The authors wish to thank Benoît Libert and Judyta Stachniak for their suggestions and ideas and Sylvie Baudine for her permanent English support.

References

1. N. Asokan, V. Shoup, and M. Waidner. Optimistic Fair Exchange of Digital Signatures. In K. Nyberg, editor, *Advances in Cryptology - Eurocrypt 98*, volume 1403 of *Lecture Notes in Computer Science*, pages 591–606. Springer-Verlag, 1998.
2. N. Asokan, V. Shoup, and M. Waidner. Optimistic Fair Exchange of Digital Signatures. *IEEE Journal on Selected Areas in Communication*, 18(4):593–610, 2000.
3. G. Ateniese. Efficient Verifiable Encryption (and Fair Exchange) of Digital Signatures. In G. Tsudik, editor, *Sixth ACM Conference on Computer and Communication Security and Privacy*, pages 138–146. ACM, November 1999.
4. G. Ateniese, J. Camenisch, M. Joye, and G. Tsudik. A Practical and Provably Secure Coalition-Resistant Group Signature Scheme. In M. Bellare, editor, *Advances in Cryptology - CRYPTO 2000*, volume 1880 of *Lecture Notes in Computer Science*, pages 255–270. Springer, 2000.
5. M. Bellare, A. Boldyreva, and S. Micali. Public-key Encryption in a Multi-User Setting: Security Proofs and Improvements. In B. Preneel, editor, *Advances in Cryptology - Eurocrypt'00*, volume 1807 of *Lecture Notes in Computer Science*. Springer, 2000.
6. M. Bellare, R. Canetti, and H. Krawczyk. A modular approach to the design and analysis of authentication and key-exchange protocols. In *Proceedings of the 30th annual Symposium on the Theory of Computing - STOC*, pages 419–428. ACM Press, 1998.
7. M. Bellare, A. Desai, D. Pointcheval, and P. Rogaway. Relations among notions of security for public-key encryption schemes. In *Advances in Cryptology - Crypto'98*, volume 1462 of *Lecture Notes in Computer Science*. Springer, 1998.

8. M. Bellare, D. Micciancio, and B. Warinschi. Foundations of group signatures: Formal definitions, simplified requirements, and a construction based on general assumptions. In E. Biham, editor, *Advances in Cryptology - EUROCRYPT 2003*, volume 2656 of *Lecture Notes in Computer Science*, pages 614–629. Springer, 2003.
9. M. Bellare and P. Rogaway. Optimal Asymmetric Encryption – How to Encrypt with RSA. In *Advances in Cryptology - Eurocrypt'94*, Lecture Notes in Computer Science. Springer, 1995.
10. M. Bellare and P. Rogaway. Provably secure session key distribution: the three party case. In *Proceedings of the 27th annual Symposium on the Theory of Computing - STOC*, pages 57–66. ACM Press, 1995.
11. D. Boneh, X. Boyen, and H. Shacham. Short Group Signatures. In *Advances in Cryptology - CRYPTO 2004*, Lecture Notes in Computer Science. Springer, 2004.
12. J. Camenisch and M. Michels. A Group Signature Scheme with Improved Efficiency. In K. Ohta and D. Pei, editors, *Advances in Cryptology - ASIACRYPT '98*, volume 1514 of *Lecture Notes in Computer Science*, pages 160–174. Springer, 1998.
13. J. Camenisch and V. Shoup. Practical Verifiable Encryption and Decryption of Discrete Logarithms. In *Advances in Cryptology - CRYPTO*, LNCS. Springer, 2003.
14. J. Camenisch and M. Stadler. Efficient Group Signature Schemes for Large Groups. In B.S. Kaliski Jr., editor, *Advances in Cryptology - CRYPTO '97*, volume 1294 of *Lecture Notes in Computer Science*, pages 410–424. Springer, 1997.
15. H. Chabbane, D.H. Phan, and D. Pointcheval. Public Traceability in Traitor Tracing Schemes. In R. Cramer, editor, *Advances in Cryptology - Eurocrypt'05*, volume 3494 of *LNCS*. Springer, 2005.
16. L. Chen, C. Kudla, and K.G. Paterson. Concurrent Signatures. In C. Cachin and J. Camenisch, editors, *Advances in Cryptology - EUROCRYPT 2004*, volume 3027 of *Lecture Notes in Computer Science*, pages 287–305. Springer, 2004.
17. B. Chor, A. Fiat, and M. Naor. Tracing traitor. In Y. Desmedt, editor, *Advances in Cryptology - Crypto'94*, volume 839 of *LNCS*, pages 257–270. Springer, 1994.
18. B. Chor, A. Fiat, M. Naor, and B. Pinkas. Tracing traitor. *IEEE Transaction on Information Theory*, 46(3):893–910, 2000.
19. R. Cramer and V. Shoup. A practical public key cryptosystem provably secure against adaptive chosen ciphertext attack. In *Advances in Cryptology - Crypto'98*, Lecture Notes in Computer Science. Springer, 1998.
20. Y. Dodis and L. Reyzin. Breaking and Repairing Optimistic Fair Exchange from PODC 2003. In M. Yung, editor, *ACM Workshop on Digital Rights Management (DRM)*, pages 47–74, 2003.
21. A. Fiat and A. Shamir. How to prove yourself : practical solutions of identification and signature problems. In G. Brassard, editor, *Advances in Cryptology - Proceedings of CRYPTO '86*, volume 263 of *Lecture Notes in Computer Science*, pages 186–194. Springer-Verlag, 1987.
22. E. Fujisaki, T. Okamoto, D. Pointcheval, and J. Stern. RSA-OAEP is Secure under the RSA Assumption. Available on eprint archive – <http://eprint.iacr.org/2000/061>.
23. M. Jakobsson and D. Pointcheval. Mutual authentication and key-exchange protocols for low power devices. In *Financial Cryptography*, pages 178–195. Springer, 2001.
24. A. Kiayias and M. Yung. Traitor tracing with constant transmission rate. In *Advances in Cryptology - Eurocrypt'02*, volume 2332 of *LNCS*, pages 450–465. Springer, 2002.

25. J. Kilian and E. Petrank. Identity Escrow. In *Advances in Cryptology - CRYPTO '98*, volume 1642 of *Lecture Notes in Computer Science*, pages 169–185. Springer, 1998.
26. O. Pandey, J. Cathalo, and J.-J. Quisquater. Fair Identification, 2005. Full version available at <http://www.cs.ucla.edu/~omkant>.
27. C. P. Schnorr. Efficient Signature Generation by Smart Cards. *Journal of Cryptology*, 4(3):161–174, 1991.
28. A. Shamir and Y. Tauman. Improved Online/Offline Signature Schemes. In *Advances in Cryptology - CRYPTO 2001*, volume 2139 of *Lecture Notes in Computer Science*, pages 355–367. Springer, 2001.
29. V. Shoup. OAEP Reconsidered. Available on eprint archive – <http://eprint.iacr.org/2000/060>.