Concurrent Signatures

Liqun Chen[1], Caroline Kudla[2*], and Kenneth G. Paterson[2**]

[1] Hewlett-Packard Laboratories, Bristol, UK
liqun.chen@hp.com
[2] Information Security Group
Royal Holloway, University of London, UK
{c.j.kudla, kenny.paterson}@rhul.ac.uk

Abstract. We introduce the concept of *concurrent signatures*. These allow two entities to produce two signatures in such a way that, from the point of view of any third party, both signatures are ambiguous with respect to the identity of the signing party until an extra piece of information (the keystone) is released by one of the parties. Upon release of the keystone, both signatures become binding to their true signers concurrently.
Concurrent signatures fall just short of providing a full solution to the problem of fair exchange of signatures, but we discuss some applications in which concurrent signatures suffice. Concurrent signatures are highly efficient and require neither a trusted arbitrator nor a high degree of interaction between parties. We provide a model of security for concurrent signatures, and a concrete scheme which we prove secure in the random oracle model under the discrete logarithm assumption.

Keywords: Concurrent signatures, fair exchange, Schnorr signatures, ring signatures.

1 Introduction

The problem of fair exchange of signatures is a fundamental and well-studied problem in cryptography, with potential application in a wide range of scenarios in which the parties involved are mutually distrustful. Ideally, we would like the exchange of signatures to be done in a *fair* way, so that by engaging in a protocol, either each party obtains the other's signature, or neither party does. It should not be possible for one party to terminate the protocol at some stage leaving the other party committed when they themselves are not.

The literature contains essentially two different approaches to solving the problem of fair exchange of signatures.

Early work on solving the problem was based on the idea of timed release or timed fair exchange of signatures [BN00,EGL85,G83]. Here, the two parties sign their respective messages and exchange their signatures "little-by-little" using

* This author is funded by Hewlett-Packard Laboratories.
** This author supported by the Nuffield Foundation NUF-NAL02.

a protocol. Typically, such protocols are highly interactive with many message flows. Moreover, one party, say B, may often be at an advantage in that he sometimes has (at least) one more bit of A's signature than she has of B's. This may not be a significant issue if the computing power of the two parties are roughly equivalent. But if B has superior computing resources, this may put him at a significant advantage since he may terminate the protocol early and use his resources to compute the remainder of A's signature, while it may be infeasible for A to do the same. Even if the fairness of such protocols could be guaranteed, they may still be too interactive for many applications. See [GP03] for further details and references for such protocols.

An alternative approach to solving the problem of fair exchange of signatures involves the use of a (semi-trusted) third party or arbitrator T who can be called upon to handle disputes between signers. The idea is that A registers her public key with T in a one-time registration, and thereafter may perform many fair exchanges with other entities. To take part in a fair exchange with B, A creates a partial signature which she sends to B. Entity B can be convinced that the partial signature is valid (perhaps via a protocol interaction with A) and that T can extract a full, binding signature from the partial signature. However, the partial signature on its own is not binding for A. B then fulfils his commitment by sending A his signature, and if valid, A releases the full version of her signature to B. The protocol is fair since if B does not sign, then A's partial signature is worthless to B, and if B does sign but A refuses to release her full signature then B can obtain it from T. The third party is only required in case of dispute; for this reason, protocols of this type are commonly referred to as optimistic fair exchange protocols. See [ASW98,ASW00,BGLS03,BW00,CS03,DR03,GJM99,PCS03] for further details of such schemes.

The main problem with such an approach is the requirement for a dispute-resolving third party with functions beyond those required of a normal Certification Authority. In general, appropriate third parties may not be available.

It is our thesis that the *full* power of fair exchange is not necessary in many application scenarios. This paper introduces a somewhat weaker concept, which we name *concurrent signatures*. The cost of concurrent signatures is that they do not provide the full security guarantees of a fair exchange protocol. Their benefit is that they have none of the disadvantages of previous solutions: they do not require a special trusted third party[3], and they do not rely on a computational balance between the parties. Moreover, our concrete realization is computationally and bandwidth efficient. Informally, concurrent signatures appear to be as close to fair exchange as it's possible to get whilst staying truly practical and not relying on special third parties.

1.1 Our Contributions

We introduce the notion of *concurrent signatures* and *concurrent signature protocols*. In a concurrent signature protocol, two parties A and B interact without

[3] Our concurrent signatures will still require a conventional CA for the distribution of public keys, but not a trusted third party with any other special functions.

the help of a third party to sign (possibly identical) messages M_A and M_B in such a way that both A and B become publicly committed to their respective messages at the same moment in time (i.e. concurrently). This moment is determined by one of the parties through the release of an extra piece of information k which we call a *keystone*. Before the keystone's release, neither party is publicly committed through their signatures, while after this point, both are. In fact, from a third party's point of view, before the keystone is released, both parties could have produced both signatures, so the signatures are completely ambiguous.

Note that the party who controls the keystone k has a degree of extra power: it controls the timing of the keystone release and indeed whether the keystone is released at all. Upon receipt of B's signature σ_B, A might privately show σ_B and k to a third party C and gain some advantage from doing so. This is the main feature that distinguishes concurrent signatures from fair exchange schemes. In a fair exchange scheme, each signer A should either have recourse to a third party to release the other party B's signature or be assured that the B cannot compute A's signature significantly more easily than A can compute B's. With concurrent signatures, only when A releases the keystone do both signatures become simultaneously binding, and there is no guarantee that A will do so. However, in the real world, there are often existing mechanisms that can naturally be used to guarantee that B will receive the keystone should his signature be used. These existing mechanisms can provide a more natural dispute resolution process than reliance on a special trusted party. We argue that concurrent signatures are suited to any fair exchange application where:

- There is no sense in A withholding the keystone because she needs it to obtain a service from B. For example, suppose B sells computers. A signs a payment instruction to pay B the price of a computer, and B signs that he authorizes her to pick one up from the depot (B's signature may be thought of as a receipt). Then A can withhold the keystone, but as soon as she tries to pick up her computer, B will ask for a copy of his signature authorizing her to collect one. In this way B can obtain the keystone which validates A's payment signature. In this example, the application itself forces the delivery of the keystone to B.
- There is no possibility of A keeping B's signature private in the long term. For example, consider the routine "four corner" credit card payment model. Here C may be A's acquiring bank, and B's signature may represent a payment to A that A must channel via C to obtain payment. Bank C would then communicate with B's issuing bank D to obtain payment against B's signature and D could ensure that B's signature, complete with keystone, reaches B (perhaps via a credit card statement). As soon as B has the keystone, A becomes bound to her signature. In this application, the back-end banking system provides a mechanism by which keystones would reach B if A were to withhold them.
- There is a single third party C who verifies both A and B's signature. Now, if A tries to present B's signature along with k to C whilst withholding k from

B, B will be able to present A's signature to C and have it verified. As an application, consider the (perhaps somewhat artificial) scenario where A and B are two politicians from different parties who want to form a coalition to jointly release a piece of information to the press C in such a way that neither of them is identified as being the sole signatory to the release. Concurrent signatures seem just right for this task. Here the keystone is not necessarily returned to B, but it does reach the third party to whom B wishes to show A's signature.

We also consider an example where concurrent signatures provide a novel solution to an old problem: that of fair tendering of contracts (our signatures can also be used in a similar way in auction applications). Suppose that A has a bridge-building contract that she wishes to put out to tender, and suppose companies B and C wish to put in proposals to win the contract and build the bridge. This process is sometimes open to abuse by A since she can privately show B's signed proposal to C to enable C to better the proposal. Using concurrent signatures, B would sign his proposal to build the bridge for an amount X, but keep the keystone private. If A wishes to accept the proposal, she returns a payment instruction to pay B amount X. She knows that if B attempts to collect the payment, then A will obtain the keystone through the banking system. But A may also wish to examine C's proposal before deciding which to accept. However there is no advantage for A to show B's signature to C since at this point B's signature is ambiguous and so C will not be convinced of anything at all by seeing it. We see that the tendering process is immune to abuse by A. We note that this example makes use of the ambiguity of our signatures prior to the keystone release, and although the solution can be realized by using standard fair exchange protocols, such protocols do not appear to previously have been suggested for this purpose.

Our schemes are not abuse-free in the sense of [BW00,GJM99], since the party A who holds the keystone can always determine whether to complete or abort the exchange of signatures, and can demonstrate this by showing an outside party C the signature from B with the keystone before revealing the keystone to B. However the above example shows that abuse can be addressed by our schemes in certain applications.

1.2 Technical Approach

We briefly explain how a concurrent signature protocol can be built using the *ambiguity* property enjoyed by ring signatures [RST01,AOS02] and designated verifier signatures [JSI96]. This introduces the key technical idea of our paper.

A two-party ring signature has the property that it could have been produced by either of the two parties. A similar property is shared by designated verifier signatures. We will refer to any signature scheme with this property as an *ambiguous* signature scheme and we will formalize the notion of ambiguity for signatures in the sequel. Since either of two parties could have produced such an ambiguous signature, both parties can deny having produced it. However, we note that if A creates an ambiguous signature which only either A or B could have created, and sends this to B, then B is convinced of the authorship of the signature (since he knows that he did not create it himself). However B cannot prove this to a third party. The same situation applies when the roles of A and B are reversed.

Suppose now that the ambiguous signature scheme has the property that, when A computes an ambiguous signature, she must choose some random bits h_B to combine with B's public key, but that the signing process is otherwise deterministic. Likewise, suppose the same is true for B with random bits h_A (when the roles of A and B are interchanged). Suppose A creates an ambiguous signature σ_A on M_A using bits h_B that are derived by applying a hash function to a string k that is secret to A; h_B is then a commitment to k. B can verify that A created the signature σ_A but not demonstrate this to a third party. Now B can create an ambiguous signature σ_B on M_B using as its input h_A the same h_B that A used. Again, A can verify that B is the signer. As long as k remains secret, neither party can demonstrate authorship to a third party.

But now if A publishes the *keystone* k, then any third party can be convinced of the authorship of both signatures. The reason for this is that the only way that B could produce σ_B is by following his signing algorithm, choosing randomness h_A and deterministically producing σ_B. The existence of a pre-image k of B's randomness h_A determines B as being the only party who could have conducted the signature generation process to produce σ_B. The same holds true for A and σ_A. Thus the pairs $\langle k, \sigma_A \rangle$ and $\langle k, \sigma_B \rangle$ amount to a simultaneously binding pair of signatures on A and B's messages. We call these pairs *concurrent* signatures.

We point out that Rivest *et al.* in their pioneering work on ring signatures [RST01] considered the situation in which an anonymous signer A wants to have the option of later proving his authorship of a ring signature. Their solution was to choose the bits h_B pseudo-randomly and later to reveal the seed used to generate h_B. In this work, we use the same trick for a new purpose: to ensure that either both or neither of the parties can be identified as signers of messages.

We note that any suitably ambiguous signature scheme can be used to produce a concurrent signature protocol. We choose to base our concrete scheme on the non-separable ring signature scheme of [AOS02]. This scheme is, in turn, an adaptation of the Schnorr signature scheme. A second concrete scheme can be built from the short ring signature scheme of [BGLS03] using our ideas. An earlier version of our scheme used the designated verifier signatures of [JSI96] instead, however it achieved slightly weaker ambiguity properties than our concrete scheme.

We give generic definitions of concurrent signatures and concurrent signature protocols, a suitably powerful multi-party adversarial model for this setting, and give a formal definition of what it means for such schemes and protocols to be secure. Security is defined via the notions of unforgeability, ambiguity and fairness.

Because our concrete scheme is ultimately based on the Schnorr signature scheme [S91], we are able to directly relate its security to the hardness of the discrete logarithm problem in an appropriate group. In doing this, we make use of the forking lemma methodology of [PS96,PS00]; for this reason, our security proof will be in the random oracle model.

2 Formal Definitions

2.1 Concurrent Signature Algorithms

We now give a more formal definition of a concurrent signature scheme. Our protocols are naturally multi-party ones, so our model assumes a system with a number of different participants that is polynomial in the security parameter l.

Definition 1. *A concurrent signature scheme is a digital signature scheme comprised of the following algorithms:*

SETUP: *A probabilistic algorithm that on input a security parameter l, outputs descriptions of: the set of participants $\mathcal{U}$, the message space $\mathcal{M}$, the signature space $\mathcal{S}$, the keystone space $\mathcal{K}$, the keystone fix space $\mathcal{F}$, and a function $KGEN : \mathcal{K} \rightarrow \mathcal{F}$. The algorithm also outputs the public keys $\{X_i\}$ of all the participants, each participant retaining their private key x_i, and any additional system parameters π.*

ASIGN: *A probabilistic algorithm that on inputs $\langle X_i, X_j, x_i, h_2, M\rangle$, where $h_2 \in \mathcal{F}$, X_i and $X_j \neq X_i$ are public keys, x_i is the private key corresponding to X_i, and $M \in \mathcal{M}$, outputs an* ambiguous signature *$\sigma = \langle s, h_1, h_2\rangle$ on M, where $s \in \mathcal{S}$, $h_1, h_2 \in \mathcal{F}$.*

AVERIFY: *An algorithm which takes as input $S = \langle \sigma, X_i, X_j, M\rangle$, where $\sigma = \langle s, h_1, h_2\rangle$, $s \in \mathcal{S}$, $h_1, h_2 \in \mathcal{F}$, X_i and X_j are public keys, and $M \in \mathcal{M}$, outputs accept or reject. We also require that if $\sigma' = \langle s, h_2, h_1\rangle$, then $AVERIFY(\sigma', X_j, X_i, M) = AVERIFY(\sigma, X_i, X_j, M)$. We call this the* symmetry *property of AVERIFY.*

VERIFY: *An algorithm which takes as input $\langle k, S\rangle$ where $k \in \mathcal{K}$ is a keystone and S is of the form $S = \langle \sigma, X_i, X_j, M\rangle$, where $\sigma = \langle s, h_1, h_2\rangle$ with $s \in \mathcal{S}$, $h_1, h_2 \in \mathcal{F}$, X_i and X_j are public keys, and $M \in \mathcal{M}$. The algorithm checks if KGEN(k)= h_2. If not, it terminates with output reject. Otherwise it runs AVERIFY(S) (in which case the output of VERIFY is just that of AVERIFY).*

We call a signature σ an *ambiguous signature* and any pair $\langle k, \sigma\rangle$, where k is a valid keystone for σ, a *concurrent signature*. The obvious correctness properties for ambiguous and concurrent signatures are formalized in Section 3.

2.2 Concurrent Signature Protocol

We will describe a concurrent signature protocol between two parties A and B (or Alice and Bob). Since one party needs to create the keystone and send the first ambiguous signature, we call this party the *initial signer*. A party who responds to this initial signature by creating another ambiguous signature with the same keystone fix we call a *matching signer*. Without loss of generality, we assume A to be the initial signer, and B the matching signer. From here on, we will use subscripts A and B to indicate initial signer A and matching signer B. The signature protocol works as follows:

A and B run SETUP to determine the public parameters of the scheme. We assume that A's public and private keys are X_A and x_A, and B's public and private keys are X_B and x_B.

1: A picks a random keystone $k \in \mathcal{K}$, and computes $f=$ KGEN(k). A takes her own public key X_A and B's public key X_B and picks a message $M_A \in \mathcal{M}$ to sign. A then computes her ambiguous signature to be

$$\sigma_A = \langle s_A, h_A, f \rangle = \text{ASIGN}(X_A, X_B, x_A, f, M_A),$$

and sends this to B.

2: Upon receiving A's ambiguous signature σ_A, B verifies the signature by checking that AVERIFY$(\langle s_A, h_A, f\rangle, X_A, X_B, M_A)=$ accept. If not B aborts, otherwise B picks a message $M_B \in \mathcal{M}$ to sign and computes his ambiguous signature

$$\sigma_B = \langle s_B, h_B, f \rangle = \text{ASIGN}(X_B, X_A, x_B, f, M_B)$$

and sends this back to A. Note that B uses the same value f in his signature as A did to produce σ_A.

3: Upon receiving B's signature σ_B, A verifies that AVERIFY$(\langle s_B, h_B, f\rangle, X_B, X_A, M_B) =$ accept, where f is the same keystone fix as A used in Step 1. If not, A aborts, otherwise A sends keystone k to B.

Note that inputs $\langle k, S_A\rangle$ and $\langle k, S_B\rangle$ will now both be accepted by VERIFY, where $S_A = \langle\langle s_A, h_A, f\rangle, X_A, X_B, M_A\rangle$ and $S_B = \langle\langle s_B, h_B, f\rangle, X_B, X_A, M_B\rangle$.

3 Formal Security Model

We present a formal security model for concurrent signatures in this section.

3.1 Correctness

We give a formal definition of correctness for a concurrent signature scheme.

Definition 2. *We say that a concurrent signature scheme is* correct *if the following conditions hold.*

If $\sigma = \langle s, h_1, f\rangle = ASIGN(X_i, X_j, x_i, f, M)$, and $S = \langle \sigma, X_i, X_j, M\rangle$, then AVERIFY$(S)=$ accept. Moreover, if KGEN$(k) = f$ for some $k \in \mathcal{K}$, then VERIFY$(k, S)=$ accept.

3.2 Unforgeability

We give a formal definition of existential unforgeability of a concurrent signature scheme under a chosen message attack in the multi-party setting. To do this, we extend the definition of existential unforgeability against a chosen message attack of [GMR88] to the multi-party setting. Our extension is similar to that of [B03] and is strong enough to capture an adversary who can simulate and observe concurrent signature protocol runs between any pair of participants. It is defined using the following game between an adversary E and a challenger C.

Setup: C runs SETUP for a given security parameter l to obtain descriptions of $\mathcal{U}$, $\mathcal{M}$, $\mathcal{S}$, $\mathcal{K}$, $\mathcal{F}$, and KGEN : $\mathcal{K} \to \mathcal{F}$. SETUP also outputs the public and private keys $\{X_i\}$ and $\{x_i\}$ and any additional public parameters π. E is given all the public parameters and the public keys $\{X_i\}$ of all participants. C retains the private keys $\{x_i\}$.

E can make the following types of query to the challenger C:

KGen Queries: E can request that C select a keystone $k \in \mathcal{K}$ and return the keystone fix $f = \text{KGEN}(k)$. If E wishes to choose his own keystone, then he can compute his own keystone fix using KGEN directly.

KReveal Queries: E can request that C reveal the keystone k that it used to produce a keystone fix $f \in \mathcal{F}$ in a previous KGEN query. If f was not a previous KGEN output then C outputs invalid, otherwise C outputs k where $f = \text{KGEN}(k)$.

ASign Queries: E can request an ambiguous signature for any input of the form $\langle X_i, X_j, h_2, M\rangle$ where $h_2 \in \mathcal{F}$, X_i and $X_j \neq X_i$ are public keys and $M \in \mathcal{M}$. C responds with an ambiguous signature $\sigma = \langle s, h_1, h_2\rangle = \text{ASIGN}(X_i, X_j, x_i, h_2, M)$. Note that using ASign queries in conjunction with KGen queries, E can obtain concurrent signatures $\langle k, \sigma\rangle$ for messages and pairs of users of his choice.

AVerify and Verify Queries: Answers to these queries are not provided by C since E can compute them for himself using the AVERIFY and VERIFY algorithms.

Private Key Extract Queries: E can request the private key corresponding to the public key of any participant X_i. In response C outputs x_i.

Output: Finally E outputs a tuple $\sigma = \langle s, h_1, f\rangle$ where $s \in \mathcal{S}$, $h_1, f \in \mathcal{F}$, along with public keys X_c and X_d, and a message $M \in \mathcal{M}$. The adversary wins the game if AVERIFY($\langle s, h_1, f\rangle, X_c, X_d, M$)= accept, and if either of the following two cases hold:

1. No ASign query with input either of the tuples $\langle X_c, X_d, f, M\rangle$ or $\langle X_d, X_c, h_1, M\rangle$ was made by E, and no Private Key Extract query was made by E on either X_c or X_d.
2. No ASign query with input $\langle X_c, X_i, f, M\rangle$ was made by E for any $X_i \neq X_c, X_i \in \mathcal{U}$, no Private Key Extract query with input X_c was made by E, and either f was a previous output from a KGen query or E produces a keystone k such that $f = \text{KGEN}(k)$.

Definition 3. *We say that a concurrent signature scheme is* existentially unforgeable under a chosen message attack in the multi-party model *if the probability of success of any polynomially bounded adversary in the above game is negligible (as a function of the security parameter l).*

Case 1 of the output conditions in the above game models forgery of an ambiguous signature in the situation where the adversary does not have knowledge of either of the respective private keys. This condition is required for our protocol so that the matching signer B is convinced that A's ambiguous signature can only originate from A. Case 2 models forgery in the situation where the adversary knows one of the private keys and so applies to the situation in our protocol where one of the two parties attempts to cheat the other. More specifically, it covers attacks where an initial signer forges a concurrent signature by a matching signer, and where a matching signer has access to an initial signer's ambiguous signature and keystone fix (but not the actual keystone) and forges a concurrent signature of the initial signer.

A further point to note is that in case 2, we insist that no ASign query of the form $\langle X_c, X_i, f, M\rangle$ is made, for any $X_i \neq X_c, X_i \in \mathcal{U}$. This is because, given a valid ambiguous signature $\sigma = \langle s, h_1, f\rangle$ for public keys X_c and X_i, and the private keys of both X_i and X_d, it may be possible to create a valid ambiguous signature $\sigma' = \langle s', h_1, f\rangle$ with public keys X_c and X_d on a message M. This is certainly the case for our concrete scheme, but should not be considered as a useful forgery because an attacker does not succeed in changing who is actually bound by the signature: in this case X_c.

3.3 Ambiguity

Ambiguity for a concurrent signature is defined by the following game between an adversary E and a challenger C.

Setup: This is as before in the game of Section 3.2.

Phase 1: E makes a sequence of KGen, KReveal, ASign and Private Key Extract queries. These are answered by C as in the unforgeability game of Section 3.2.

Challenge: Then E selects a challenge tuple $\langle X_i, X_j, M\rangle$ where X_i and X_j are public keys, and $M \in \mathcal{M}$ is the message to be signed. In response, C randomly selects $k \in \mathcal{K}$ and computes $f = \mathrm{KGEN}(k)$, then randomly selects a bit $b \in \{0,1\}$. C outputs $\sigma_1 = \langle s_1, h_1, f\rangle = \mathrm{ASIGN}(X_i, X_j, x_i, f, M)$ if $b = 0$; otherwise C computes $\sigma_2' = \langle s_2, h_2, f\rangle = \mathrm{ASIGN}(X_j, X_i, x_j, f, M)$ and outputs $\sigma_2 = \langle s_2, f, h_2\rangle$.

Phase 2: E may make another sequence of queries as in Phase 1; these are handled by C as before.

Output: Finally E outputs a guess bit $b' \in \{0,1\}$. E wins if $b' = b$ and E has not made a KReveal query on any of the values f, h_1 or h_2.

Definition 4. *We say that a concurrent signature scheme is* ambiguous *if no polynomially bounded adversary has advantage that is non-negligibly greater than $1/2$ of winning in the above game.*

We note that ambiguity in our concrete concurrent signature scheme will come directly from the ambiguity property of an underlying ring signature scheme. However the definition for ambiguity (or anonymity) in two-party ring signatures [RST01,BSS02,ZK02] states that an unbounded adversary should have probability exactly $1/2$ of guessing b correctly. Our definition must be slightly weaker because in our ambiguous signatures, one of our h values is generated by KGEN and is therefore at best pseudorandom. However, since we model KGEN by a random oracle when proving ambiguity for our concrete scheme, we achieve perfect ambiguity as in the stronger definition for ring signatures.

3.4 Fairness

We require the concurrent signature scheme and protocol to be fair for both an initial signer A, and a matching signer B. This concept is defined via the following game between an adversary E and a challenger C:

Setup: This is as before in the game of Section 3.2.

KGen, KReveal, ASign and Private Key Extract Queries: These queries are answered by C as in the unforgeability game of Section 3.2.

Output: Finally E chooses the challenge public keys X_c and X_d, outputs a keystone $k \in \mathcal{K}$, and $S = \langle \sigma, X_c, X_d, M \rangle$ where $\sigma = \langle s, h_1, f \rangle$, $s \in \mathcal{S}$, $h_1, f \in \mathcal{F}$, and $M \in \mathcal{M}$, and where AVERIFY(S) = accept. The adversary wins the game if either of the following cases hold:

1. If f was a previous output from a KGen query, no KReveal query on input f was made, and if $\langle k, S \rangle$ is accepted by VERIFY.
2. If E also produces $S' = \langle \sigma', X_d, X_c, M' \rangle$, with $\sigma' = \langle s', h_1', f \rangle$, $s' \in \mathcal{S}$, $h_1', f \in \mathcal{F}$, message $M' \in \mathcal{M}$, where AVERIFY(S') = accept, and $\langle k, S \rangle$ is accepted by VERIFY, but $\langle k, S' \rangle$ is not accepted by VERIFY.

Definition 5. *We say that a concurrent signature scheme is* fair *if a polynomially bounded adversary's probability of success in the above game is negligible.*

Our definition of fairness formalizes our intuitive understanding of fairness for A in the protocol of Section 2.2 (in case 1 of the output conditions), since it guarantees that only the entity who generates a keystone can use it to create a binding signature (by revealing it). It also captures fairness for B (in case 2 of the output conditions), since it guarantees that any valid ambiguous signatures produced using the same keystone fix will all become binding. Thus B cannot be left in a position where a keystone binds his signature to him while A's initial signature is not also bound to A. However note that our definition does not guarantee that B will ever receive the necessary keystone.

3.5 Security

Definition 6. *We say that a correct concurrent signature scheme is* secure *if it is existentially unforgeable under a chosen message attack in the multi-party setting, ambiguous, and fair.*

4 A Concrete Concurrent Signature Scheme

We present a concrete concurrent signature scheme in which the underlying ambiguous signatures and the resulting concurrent signatures are obtained by modifying signatures in the basic scheme of Schnorr [S91]. The scheme's algorithms (SETUP, ASIGN, AVERIFY, VERIFY) are as follows:

SETUP: On input a security parameter l, two large primes p and q are selected such that $q|p-1$. These are published along with an element g of $(\mathbb{Z}/p\mathbb{Z})^*$ of order q, where q is exponential in l. The spaces $\mathcal{S}, \mathcal{F}, \mathcal{M}, \mathcal{K}$ are defined as follows: $\mathcal{S} \equiv \mathcal{F}=\mathbb{Z}_q$ and $\mathcal{M} \equiv \mathcal{K}=\{0,1\}^*$. Two cryptographic hash functions $H_1, H_2 : \{0,1\}^* \to \mathbb{Z}_q$ are also selected and we define KGEN to be H_1. Private keys $x_i, 1 \leq i \leq n$ are chosen uniformly at random from $\mathbb{Z}_q$, where n is polynomial in l. The public keys are computed as $X_i = g^{x_i} \bmod p$ and are made public.

ASIGN: This algorithm takes as input $\langle X_i, X_j, x_i, h_2, M\rangle$, where $X_i, X_j \neq X_i$ are public keys, $x_i \in \mathbb{Z}_q$ is the private key corresponding to X_i, $h_2 \in \mathcal{F}$ and $M \in \mathcal{M}$ is a message. The algorithm picks a random value $t \in \mathbb{Z}_q$ and then computes the values:

$$\begin{aligned} h &= H_2(g^t {X_j}^{h_2} \bmod p || M), \\ h_1 &= h - h_2 \bmod q, \\ s &= t - h_1 x_i \bmod q. \end{aligned}$$

Here "||" denotes concatenation. The algorithm outputs $\sigma = \langle s, h_1, h_2\rangle$.

AVERIFY: This algorithm takes as input $\langle \sigma, X_i, X_j, M\rangle$ where $\sigma = \langle s, h_1, h_2\rangle$, $s \in \mathcal{S}$, $h_1, h_2 \in \mathcal{F}$, X_i and X_j are public keys, and $M \in \mathcal{M}$ is a message. The algorithm checks that the equation

$$h_1 + h_2 = H_2(g^s X_i^{h_1} X_j^{h_2} \bmod p \; || M) \quad \bmod q$$

holds, and if so, outputs accept. Otherwise, it outputs reject.

VERIFY: This algorithm is defined in terms of KGEN and AVERIFY, as in Section 2.1.

The ASIGN algorithm is a direct modification of the ring signature algorithm of [AOS02], and guarantees our property of ambiguity before the keystone is revealed. We require that $X_j \neq X_i$ since otherwise the signature would be a standard Schnorr signature [S91] and would not be ambiguous. It is also easily checked that the scheme satisfies the definition of correctness and that AVERIFY has the required symmetry property.

A concrete concurrent signature protocol can be derived directly from the algorithms defined above and the generic protocol in Section 2.2.

5 Security of the Concrete Concurrent Signature Scheme

We now state some security results for the concrete scheme of Section 4. The proofs of Lemmas 1 and 3 are proved in Appendix A. The proof of Lemma 2 is routine, and the details are left to the reader. Our proofs of security are in the random oracle model [BR93].

Lemma 1. *The concurrent signature scheme of Section 4 is existentially unforgeable under a chosen message attack in the random oracle model, assuming the hardness of the discrete logarithm problem.*

Lemma 2. *The concurrent signature scheme of Section 4 is ambiguous in the random oracle model.*

Lemma 3. *The concurrent signature scheme of Section 4 is fair in the random oracle model.*

Theorem 1. *The concurrent signature scheme of Section 4 is secure in the random oracle model, assuming the hardness of the discrete logarithm problem.*

Proof. The proof follows directly from Lemmas 1, 2 and 3. □

6 Extensions and Open Problems

6.1 The Scheme Can Use a Variety of Keys

Our concurrent signature scheme can be based on any ring signature scheme, as long as it is compatible with the keystone fix idea. Thus it is feasible to build concrete concurrent signature schemes using a variety of key types, and therefore the security of such schemes may be based on a variety of underlying hard problems. Furthermore, the key pairs in a single concurrent signature scheme do not have to be of the same type. The techniques to be used for achieving concurrent signatures from a variety of keys are the same as the key separability techniques for ring signatures as described in [AOS02].

6.2 The Multi-party Case

It would be interesting to see if concurrent signatures could be extended to the multi-party case, that is, where many entities can fairly exchange signatures concurrently. The existing two party scheme can trivially be extended to include multiple matching signers. However we do not as yet have a model for fairness for such a scheme. It would also be interesting to investigate methods whereby the revelation of keystones did not depend entirely on the initial signer, but on the other signing parties as well.

7 Conclusion

We introduced the notion of concurrent signatures, presented a concurrent signature scheme and related its security to the hardness of the discrete logarithm problem in an appropriate security model. We have also discussed some applications for concurrent signatures, and the advantages they have over previous work. In particular, we have compared concurrent signatures to techniques for fair exchange of signatures, and presented some applications in which the full security of fair exchange may not be necessary and the more pragmatic solution of concurrent signatures suffice.

Acknowledgements

We would like to thank Alex Dent for useful comments on the paper. We also thank Keith Harrison, who suggested the name "keystone", and Brian Monahan for some useful discussions.

References

[AOS02] M. Abe, M. Ohkubo, and K. Suzuki, 1-out-of-n signatures from a variety of keys, In *Advances in Cryptology - ASIACRYPT 2002*, LNCS vol. 2501, pp. 415-432. Springer-Verlag, 2002.

[ASW98] N. Asokan, V. Shoup, and M. Waidner, Optimistic fair exchange of signatures, In *Advances in Cryptology - EUROCRYPT 1998*, LNCS vol. 1403, pp. 591-606. Springer-Verlag, 1998.

[ASW00] N. Asokan, V. Shoup, and M. Waidner, Optimistic fair exchange of signatures, In *IEEE Journal on Selected Areas in Communication* vol. 18(4), pp. 593-610, 2000.

[B03] X. Boyen, Multipurpose identity-based signcryption. A Swiss army knife for identity-based cryptography. In *Advances in Cryptology - CRYPTO 2003*, LNCS vol. 2729, pp. 383-399, Springer-Verlag, 2003.

[BGLS03] D. Boneh, C. Gentry, B. Lynn and H. Shacham, Aggregrate and verifiably encrypted signatures from bilinear maps. In *Advances in Cryptology - EUROCRYPT 2003*, LNCS vol. 2656, pp. 416-432, Springer-Verlag, 2003.

[BN00] D. Boneh, and M. Naor, Timed commitments (extended abstract). In *Advances in Cryptology - CRYPTO 2000*, LNCS vol. 1880, pp. 236-254. Springer-Verlag, 2000.

[BR93] M. Bellare, and P. Rogaway, Random oracles are practical: a paradigm for designing efficient protocols. In *Proc. of the 1st CCCS*, pp. 62-73. ACM press, 1993.

[BSS02] E. Bresson, J. Stern and M. Szydlo, Threshold ring signatures for ad-hoc groups. In *Advances in Cryptology - CRYPTO 2002*, LNCS vol. 2442, pp. 465-480, Springer-Verlag, 2002.

[BW00] B. Baum-Waidner and M. Waidner, Round-optimal and abuse free optimistic multi-party contract signing. In Proc. of *Automata, Languages and Programming*, pp. 524-535, 2000.

[CS03] J. Camenisch and V. Shoup, Practical verifiable encryption and decryption of discrete logarithms. In *Advances in Cryptology - CRYPTO 2003*, LNCS vol. 2729, pp. 126-144, Springer-Verlag, 2002.

[DR03] Y. Dodis, and L. Reyzin, Breaking and repairing optimistic fair exchange from PODC 2003, In *ACM Workshop on Digital Rights Management (DRM)*, October 2003.

[EGL85] S. Even, O. Goldreich, and A. Lempel, A randomized protocol for signing contracts. In *Commun. ACM*, vol. 28(6), pp. 637-647, June 1985.

[G83] O. GOLDREICH, A simple protocol for signing contracts. In *Advances in Cryptology - CRYPTO 1983*, pp. 133-136, Springer-Verlag, 1983.
[GJM99] J. GARAY, M. JAKOBSSON AND P. MACKENZIE, Abuse-free optimistic contract signing, In *Advances in Cryptology - CRYPTO 1999*, LNCS vol. 1666, pp. 449-466, Springer-Verlag, 1999.
[GMR88] S. GOLDWASSER, S. MICALI AND R. RIVEST, A digital signature scheme secure against adaptive chosen message attacks. In *SIAM Journal of Computing*, vol. 17(2), pp. 281-308, April 1988.
[GP03] J. GARAY, AND C. POMERANCE, Timed fair exchange of standard signatures, In *Proc. Financial Cryptography 2003*, LNCS vol. 2742, pp. 190-207, Springer-Verlag, 2003.
[JSI96] M. JAKOBSSON, K. SAKO, AND R. IMPAGLIAZZO, Designated verifier proofs and their applications, In *Advances in Cryptology - EUROCRYPT 1996*, LNCS Vol.1070, pp. 143-154, Springer-Verlag, 1996.
[PCS03] J. PARK, E. CHONG, AND H. SIEGEL, Constructing fair exchange protocols for e-commerce via distributed computation of RSA signatures. In *22nd Annual ACM Symp. on Principles of Distributed Computing* pp. 172-181, July 2003.
[PS96] D. POINTCHEVAL AND J. STERN, Security proofs for signature schemes, In *Advances in Cryptology - EUROCRYPT 1996*, LNCS vol. 1070, pp. 387-398, Springer-Verlag, 1996.
[PS00] D. POINTCHEVAL AND J. STERN, Security arguments for digital signatures and blind signatures. In *Journal of Cryptology*, vol. 13, pp. 361-396, 2000.
[RST01] R. RIVEST, A. SHAMIR AND Y. TAUMAN, How to leak a secret. In *Advances in Cryptology - ASIACRYPT 2001*, LNCS 2248, pp. 552-565, Springer-Verlag, 2001.
[S91] C.P. SCHNORR, Efficient signature generation by smart cards, In *Journal of Cryptology*, vol. 4, no. 3, pp. 161-174, 1991.
[ZK02] F. ZHANG AND K. KIM, ID-based blind signature and ring signature from pairings. In *Advances in Cryptology - ASIACRYPT 2002*, LNCS vol. 2501, pp. 533-547, Springer-Verlag, 2002.

Appendix A

Proof of Lemma 1. The proof is similar to the proof of unforgeability of the Schnorr signature scheme [S91] by Pointcheval and Stern [PS96], and makes use of the forking lemma [PS96,PS00].

The Forking Lemma [PS96,PS00]: The forking lemma applies in particular to signature schemes which on input a message M produce signatures of the form (r_1, h, r_2) where r_1 takes its value randomly from a large set, h is the hash of M and r_1, and r_2 depends only on r_1, M and h.

The forking lemma in [PS00] states that if E is a polynomial time Turing machine with input only public data, which produces, in time τ and with probability $\eta \geq 10(\mu_s+1)(\mu_s+\mu)/2^l$ (where l is a security parameter) a valid signature (m, r_1, h, r_2), where μ is the number of hash queries, and μ_s is the number of signature queries, and if triples r_1, m, r_2 are simulatable with indistinguishable probability distribution without knowledge of the secret key, then there exists an algorithm A, which controls E and replaces E's interaction with the signer by the simulation, and which produces two valid signatures (m, r_1, h, r_2) and (m, r_1, h', r_2') such that $h \neq h'$ in expected time at most $\tau' = 120686\mu_s\tau/\eta$.

Firstly, we note that our concurrent signature scheme in Section 4 on input a message M, public keys X_i and X_j and a value h_2, produces signatures of the required form $\langle r_1, h, r_2\rangle$, where $r_1 = g^t X_j^{h_2} \bmod p$ which takes its values randomly from $\mathbb{Z}_q$, $h = h_1 + h_2$ is the hash of M and r_1, and $r_2 = s$ depends on r_1, M and h. Although the actual output of the signature is the tuple $\langle s, h_1, h_2\rangle$, the values r_1, h and r_2 can easily be derived from the output. We also note that if by the forking methodology, we have two valid signatures (r_1, h, r_2) and (r_1, h', r_2') on the same message M with $h \neq h'$, then provided that the value h_2 is computed before the relevant H_2 query, then this would be equivalent to two concurrent signatures $\langle s, h_1, h_2\rangle$ and $\langle s', h_1', h_2\rangle$ with $h_1 \neq h_1'$.

We suppose that H_1 and H_2 are random oracles, and suppose there exists an algorithm E who is able to forge concurrent signatures. So we assume that E is an attacker that makes at most μ_i queries to the random oracles $H_i, i = \{1, 2\}$, at most μ_s queries to the signing oracle, and wins the unforgeability game of Section 3.2 in time at most τ with probability at least $\eta = 10(\mu_s + 1)(\mu_s + \mu_2)/q$, where q is exponential in security parameter l.

We show how to construct an algorithm B that uses E to solve the discrete logarithm problem. B will simulate the random oracles and the challenger C in a game with E. B's goal is to solve the discrete logarithm problem on input $\langle g, X, p, q\rangle$, that is to find $x \in \mathbb{Z}_q$ such that $g^x = X \bmod p$, where g is of prime order q modulo prime p.

Simulation: B gives the parameters $\langle g, p, q\rangle$ to E. B generates a set of participants U, where $|U| = \rho(l)$ and ρ is a polynomial function of the security parameter l. Each participant has a public key X_i and private key x_i. B guesses that E will choose X_α in the position of X_c in its output. B sets $X_\alpha = X$, and for each $i \neq \alpha$, x_i is chosen randomly from $\mathbb{Z}_q$, and B sets $X_i = g^{x_i} \bmod p$. E is given all the public keys X_i. B now simulates the challenger by simulating all the oracles which E can query as follows:

H_1-Queries: E can query the random oracle H_1 at any time. B simulates the random oracle by keeping a list of tuples $\langle M_i, r_i\rangle$ which is called the H_1-List. When the oracle is queried with an input $M \in \{0,1\}^*$, B responds as follows:
1. If the query M is already on the H_1-List in the tuple $\langle M, r_i\rangle$, then B outputs r_i.
2. Otherwise B selects a random $r \in \mathbb{Z}_q$, outputs r and adds $\langle M, r\rangle$ to the H_1-List.

H_2-Queries: E can query the random oracle H_2 at any time. B simulates the H_2 oracle in the same way as the H_1 oracle by keeping an H_2-List of tuples.

KGen Queries: E can request that the challenger select a keystone $k \in \mathcal{K}$ and return a keystone fix $f = H_1(k)$. B maintains a K-List of tuples $\langle k, f\rangle$, and answers queries by choosing a random keystone $k \in \mathcal{K}$ and computing $f = H_1(k)$. B outputs f and adds the tuple $\langle k, f\rangle$ to the K-List. Note that K-List is a sublist of H_1-List, but is required to answer KReveal queries.

KReveal Queries: E can request the keystone of any keystone fix $f \in \mathcal{F}$ produced by a previous KGen Query. If there exists a tuple $\langle k, f \rangle$ on the K-List, then B returns k, otherwise it outputs invalid.

ASign Queries: B simulates the signature oracle by accepting signature queries of the form $\langle X_i, X_j, h_2, M \rangle$ where $h_2 \in \mathcal{F}$, X_i and $X_j \neq X_i$ are public keys, and $M \in \mathcal{M}$ is the message to be signed. If $X_i \neq X_\alpha$ then B computes the signature as normal and outputs $\sigma = \langle s, h_1, h_2 \rangle = \text{ASIGN}(X_i, X_j, x_i, h_2, M)$. If $X_i = X_\alpha$ then B answers the query as follows:

1. B picks a random h_1 and s in $\mathbb{Z}_q$, computes $T = g^s X_i^{h_1} X_j^{h_2} \bmod p$, and forms the string "$T||M$".
2. If $h = h_1 + h_2$ is equal to some previous output for the H_2 oracle, or if "$T||M$" was some previous input, then return to step 1.
3. Otherwise add the tuple $\langle T||M, h \rangle$ to the H_2-List.
4. B outputs $\sigma = \langle s, h_1, h_2 \rangle$ as the signature for message M with public keys X_i and X_j.

Private Key Extract Queries: E can request the private key for any public key X_i. If $X_i = X_\alpha$, then B terminates the simulation with E having failed to guess the correct challenge public key. Otherwise B returns the appropriate private key x_i.

Output: Finally, with non-negligible probability, E outputs a signature $\sigma = \langle s, h_1, f \rangle$ where $s \in \mathcal{S}$, $h_1, f \in \mathcal{F}$, along with public keys X_c and X_d, and a message $M \in \mathcal{M}$, where AVERIFY($\langle s, h_1, f \rangle, X_c, X_d, M$)= accept, and one of the following two cases holds:

1. No ASign query with input either of the tuples $\langle X_c, X_d, f, M \rangle$ or $\langle X_d, X_c, h_1, M \rangle$ was made by E, and no Private Key Extract query was made by E on either X_c or X_d.
2. No ASign query with input $\langle X_c, X_i, f, M \rangle$ was made by E for any $X_i \neq X_c, X_i \in \mathcal{U}$, no Private Key Extract query with input X_c was made by E, and either f was a previous output from a KGen query or E produces a keystone k such that $f = \text{KGEN}(k)$.

It is easy to show that case 1 of the output conditions can occur only with negligible probability δ. This follows immediately from the unforgeability of the underlying ring signature [AOS02], assuming the hardness of the discrete logarithm problem. An outline of the ring signature unforgeability proof is given in [AOS02], hence we omit the details here. Since the adversary wins the game with non-negligible probability, we assume that case 2 must have occurred.

If $X_c \neq X_\alpha$ then B aborts, having failed to guess the correct challenge public key. Henceforth, we assume that $X_c = X_\alpha = X$ (this occurring with probability $1/\rho(l)$ where ρ is a polynomial function). Given that B does not abort for any reason, it can be seen that, because of the way B handles oracle queries, the simulation seen by E is indistinguishable from a real interaction with a challenger.

Because in case 2 algorithm AVERIFY with E's signature as input returns accept, we have the equation $h = h_1 + f = H_2(g^s X_c^{h_1} X_d^f \bmod p \, ||M)$. We now analyze two further cases.

Case 1. We recall that we can rewrite the signature above in the form (r_1, h, r_2). If $h = h_1 + f$ has never appeared in any previous signature query before, then by the forking lemma, B can repeat its simulation so that E produces another such signature (r_1, h', r_2'), with $h \neq h'$.

Note that E has in fact produced two signatures $\sigma = \langle s, h_1, f \rangle$ and $\sigma' = \langle s', h_1', f' \rangle$, with $h = h_1 + f \neq h_1' + f' = h'$. If $h_1 = h_1'$, then B aborts. However, if $h_1 = h_1'$, then the h_1 values must have been computed before the relevant H_2 queries (which produced h and h'), or h_1 and h_1' are independent of h and h' respectively. Also, if $h_1 = h_1'$, then $f \neq f'$, so these values must have been computed after the relevant H_2 queries, and satisfy the equations $f = h - h_1$ and $f' = h' - h_1'$. But we know that f is also an output of H_1, either from a direct H_1 query, or via a KGen query, and the probability that an output from H_1 query matches (some function of) an output from some H_2 query is at most $\mu_2\mu_1/q$. This is negligible, so we assume that $f = f'$, and therefore that $h_1 \neq h_1'$.

Now, since h and h' resulted from different oracle queries on the same input, we know that $g^s X^{h_1} X_d^f = g^{s'} X^{h_1'} X_d^f \mod p$. So taking the exponents from both sides we get $s + xh_1 = s' + xh_1' \mod q$. Since $h_1 \neq h_1'$, B can now solve for x, the discrete logarithm of X, using the equation $x = \frac{s-s'}{h_1'-h_1} \mod q$.

So in Case 1, the probability that B does not have to abort at some point in the simulation is at least

$$\gamma = (1-\delta) \cdot \frac{1}{\rho(l)} \cdot (1 - \frac{\mu_1\mu_2}{q}),$$

which is non-negligible in security parameter l. So B solves the discrete logarithm of X by the forking lemma, in expected time at most $\tau'/\gamma = 120686\mu_s\tau/\eta\gamma$. This contradicts the hardness of the discrete logarithm problem.

Case 2. However suppose that $h = h'$, where $h' = h_1' + f'$ was the output in some previous signature query $\langle X_{c'}, X_{d'}, f', M' \rangle$. Say the previous signature was $\sigma' = \langle s', h_1', f' \rangle$ with public keys $X_{c'}$ and $X_{d'}$ on message M'. Now $h = H_2(g^s X^{h_1} X_d^f \mod p || M)$, $h' = H_2(g^{s'} X_{c'}^{h_1'} X_{d'}^{f'} \mod p || M')$ and $h = h'$. If the inputs to H_2 are not equal, then B aborts. This occurs with probability $\mu_2\mu_s/q$. Otherwise we have that the inputs to the random oracle are equal, so $M = M'$ and $g^s X^{h_1} X_d^f = g^{s'} X_{c'}^{h_1'} X_{d'}^{f'} \mod p$.

If $X_{c'}, X_{d'} \neq X$, or $X_{c'} = X$ or $X_{d'} = X$ but their exponents are different (e.g. if $X_{c'} = X$ but $h_1' \neq h_1$), then it is easy to see that B can extract x directly from the equation $g^s X^{h_1} X_d^f = g^{s'} X_{c'}^{h_1'} X_{d'}^{f'} \mod p$.

However suppose that either $X_{c'} = X$ or $X_{d'} = X$, and their exponents are equal. If $X_{c'} = X$ and $h_1' = h_1$, then since $h_1 + f = h_1' + f'$, we have that $f' = f$. But this is impossible since it contradicts the assumption that no tuple of the form $\langle X_c, X_i, f, M \rangle$ was queried on the signing oracle before.

If $X_{d'} = X$ and $h_1 = f'$, then $f = h_1'$, where $h_1' = h' - f'$ was generated in a previous signature query, and is determined by the outputs of the random oracles H_1 and H_2. But we know that f is also a direct output of H_1, perhaps via a KGen query. However the probability that an output from H_1 matches an

h'_1 from some signature query is $\mu_1\mu_s/q$. This probability is negligible and if this case occurs, then B aborts.

So for Case 2, the probability that B is not forced to abort at some point is at least

$$\gamma = (1-\delta)\cdot\frac{1}{\rho(l)}\cdot(1-\frac{\mu_2\mu_s}{q})\cdot(1-\frac{\mu_1\mu_s}{q}),$$

which is non-negligible in security parameter l. If B is not forced to abort, then B can solve the discrete logarithm of X directly from E's output. Our analysis therefore shows that in Case 2, B can extract the discrete logarithm of X within expected time at most τ/γ. This again contradicts the hardness of the discrete logarithm problem. □

Proof of Lemma 3. We suppose that H_1 and H_2 are random oracles as before, and suppose that there exists an algorithm E that with non-negligible probability wins the game in Section 3.4. In this game, the challenger runs the SETUP algorithm to initialize all the public parameters as usual, choosing all the private keys x_i randomly from $\mathbb{Z}_q$, generating the public keys as $X_i = g^{x_i} \bmod p$, and giving these public keys to E. Also as part of this game, C responds to H_1, H_2, KGen and KReveal queries as usual, and responds to ASign and Private Key Extract queries using its knowledge of the private keys.

In the final stage of the game, E chooses challenge public keys X_c, X_d and with non-negligible probability η outputs keystone k and $S = \langle\sigma, X_c, X_d, M\rangle$ with $\sigma = \langle s, h_1, f\rangle$ for which one of the following cases holds:

1. f was a previous output from a KGen query, f was not queried on the KReveal oracle, and $\langle k, S\rangle$ is accepted by VERIFY.
2. E also produces $S' = \langle\sigma', X_d, X_c, M'\rangle$, where $\sigma' = \langle s', h'_1, f\rangle$ is an ambiguous signature on M' with public keys X_d, X_c, both S and S' are accepted by AVERIFY, $\langle k, S\rangle$ is accepted by VERIFY, but $\langle k, S'\rangle$ is not accepted by VERIFY.

We now analyse E's output.

Case 1. Suppose case 1 of the output conditions occurs. Then E has found a keystone k and an output of a KGen query f such that $f = H_1(k)$, but without making a KReveal query on input f. However, since H_1 is a random oracle, E's probability of producing such a k is at most $\mu_1\mu_2/q$, where μ_1 is the number of H_1 queries made by E and μ_2 is the number of KGen queries made by E. Since both μ_1 and μ_2 are polynomially bounded in the security parameter l and q is exponential in l, this probability is negligible. This contradicts our assumption that E wins the game with non-negligible probability.

Case 2. Suppose case 2 of the output conditions occurs. Since S is accepted by AVERIFY and $\langle k, S\rangle$ is accepted by VERIFY, we must have KGEN(k)=f. But then, since S and S' share the value f, we must also have that $\langle k, S'\rangle$ is accepted by VERIFY. This is a contradiction. □