Chameleon Signatures

Hugo Krawczyk
Dept. of EE, Technion
and IBM Research
Email: hugo@ee.technion.ac.il

Tal Rabin
IBM Research
Email: talr@watson.ibm.com

Abstract

This paper presents a new tool for enhancing the confidentiality and privacy of electronic transactions such as the signing of agreements, commitment to bids, etc. We introduce chameleon signatures *that provide with an undeniable commitment of the signer to the contents of a signed document (as regular digital signatures do) but, at the same time, do not allow the recipient of the signature to disclose the contents of the signed information to any third party without the signer's consent.*

Chameleon signatures are closely related to the much researched notion of "undeniable signatures" but they allow for simpler and more efficient realizations. In particular, they do not involve the design and complexity of zero-knowledge proofs on which traditional undeniable signature schemes are based, and they are non-interactive without requiring idealized random-oracle assumptions. Chameleon signatures are generated under the standard method of hash-then-sign, where the hash is implemented via chameleon hash functions *which are characterized by the non-standard property of being collision-resistant for the signer but collision tractable for the recipient.*

We present simple and efficient implementations of chameleon hashing and chameleon signatures. The former are constructed based on standard cryptographic assumptions (e.g., the hardness of factoring or discrete logarithms), while the signature part can use any digital signature scheme (e.g., RSA or DSS). We prove the unforgeability of the resultant chameleon signatures solely based on the unforgeability of the underlying digital signature in use.

1 Introduction

Typical business relationships between companies or individuals involve commitments assumed by the parties in the form of agreements and contracts. Digital signatures represent the main cryptographic tool to provide the non-repudiation property required in case of possible disputes. However, digital signatures also allow any party to disclose (and prove!) the other party's commitment to an outsider. This may be undesirable in many business and e-commerce situations. For example, disclosing a signed contract to a journalist or a competitor can benefit one party but jeopardize the interests of the other; early dissemination of confidential agreements can be used to achieve illegitimate earnings in the stock market; a losing bidder may want to prevent disclosure of his bid even after an auction is over. These, and many other, examples show how privacy, confidentiality and legal issues raise the need to prevent the uncontrolled dissemination of the contents of some agreements and contracts by third parties or even by the recipient of a signature. Yet, also in these cases it is essential to preserve the non-repudiation property in the event of legal disputes. In such a case, an authorized judge, or arbiter, should be able to determine the validity of a contract, an agreement or commitment.

This paper presents a practical solution to this problem which we believe will become increasingly important as large-scale electronic commerce becomes prevalent. We expect to see an increased demand for confidential signature tools as more agreements are signed remotely and the risk of massive disclosure of confidential documents (e.g. via unauthorized or malicious postings in the Internet) becomes more realistic.

Previous work has dealt with the problem of bridging between the contradictory requirements of non-repudiation and controlled dissemination via the notion of *undeniable signatures*. This no-

tion was introduced by Chaum and van Antwerpen [CA89] and followed by many research works, e.g. [Cha90, BCDP90, DY91, FOO91, Ped91, CvHP91, Cha94, Oka94, Mic96, DP96, JSI96, JY96, GKR97a]. [1] The basic paradigm behind this type of signatures is that verification of a signature requires the collaboration of the signer, so that the latter can control to whom the signed document is being disclosed. Thus, a crucial requirement is that the signature will be *non-transferable*, i.e. it will not convey any information on the contents of the signed document to anyone except for those parties that engage in some specified protocol directly with the signer. Such a protocol enables the signer to confirm a valid signature or deny an invalid one. To prevent leaking of information these protocols are based on zero-knowledge proofs. As it is natural to expect, these additional properties and techniques add to the complexity of the schemes relative to regular digital signatures, both conceptually and in terms of computational and communication costs.

Our paper presents a *much simpler implementation* of the notion of undeniable signatures at the expense of some limitation in its functionality. Yet the essence of undeniable signatures (i.e., bridging between non-repudiation and controlled disclosure) is achieved by our schemes at a significantly lower cost and complexity. We name this new tool *chameleon signatures* (the reason for this name will be clear from the presentation below). The main technical novelty of chameleon signatures is in departing from the zero-knowledge paradigm common to all previous realizations of undeniable signatures (either interactive or non-interactive). Instead, chameleon signatures are built much like regular digital signatures, by following the traditional approach of *hash-then-sign*. Chameleon signatures simply apply a regular digital signature scheme (such as RSA or DSS) to a special type of hashing called *chameleon hash functions*. The latter are a functional representation of chameleon commitment schemes [BCC88], and can be thought of as collision-resistant functions with a trapdoor for finding collisions.

The basic idea is to build the signature scheme in such a way that a signature provided by a signer S to a recipient R gives R the ability to *forge* further signatures of S at will. (That is, once R receives a signature of S on a document m he can produce signatures of S on any other document m'.) Clearly, this prevents R from *proving* the validity of S's signature to a third party as he could have produced such a signature by himself. But then what is the value of such a signature if no one can decide on its validity or invalidity? We render the scheme valuable by providing the signer S with the *exclusive* ability to prove that a forged signature is in fact a forgery. In other words, R can produce forgeries that are indistinguishable from real signatures to any third party, but S can prove the forgery to such a third party if he desires or is compelled (e.g. by force of the law) to do so. Our method is essentially non-interactive. A signature is provided as a string that can be (non-interactively) verified by the recipient, while for denying a false signature, the signer only needs to provide a short piece of information as evidence for the forgery.

We can state in short several of the advantages of our schemes that make them attractive for practical implementation.

- Full non-interactivity (chameleon signatures are functions – producing short strings – rather than full protocols)
- Compatibility with the standard signature approach of hash-then-sign, and applicability of standard signature schemes such as RSA and DSS.
- Efficiency: the computational cost is no more than twice the cost of RSA or DSS signatures, and implementation uses standard tools.
- Simplicity: our schemes enjoy both conceptual and implementation simplicity; this is achieved especially by avoiding the use of zero-knowledge proofs in the construction.
- Standard cryptographic assumptions (existence of unforgeable signature schemes, and discrete-log or factoring hardness) suffice for fully proving the security of our schemes. In particular, non-interactivity is obtained *without* recurring to idealized assumptions such as SHA-1 acting as a random oracle.
- Convertibility: this property is discussed below and has been long considered as an important advantage of those undeniable signature schemes that achieve it. We obtain it in a natural and efficient way.

CONVERTIBILITY. In the undeniable signature literature, the property of *convertibility* has received a lot of attention. This notion (introduced in [BCDP90]) represents the ability of the signer to

[1] Interestingly, this type of interactive signatures was already suggested in 1976 by Michael Rabin based on one-time signatures [Rab78].

eventually release a piece of information that converts the undeniable signature into a regular digital signature with the traditional property that anyone can verify it without the help of the signer. This can be a useful property for signatures that loose their non-transferability requirement after some time, or after some event (e.g., after a bid is assigned or a confidentiality period is over). Our schemes for chameleon signatures provide for simple ways to achieve convertibility. We present selective and total conversion techniques. The first means that individual (selected) signatures can be converted by providing some information specific to that signature. Total conversion means that the signer releases some (short) piece of information that converts *all* the signatures in a pre-specified set into regular signatures.

RELATION TO UNDENIABLE SIGNATURES. As said above, chameleon signatures are closely related to the notion of undeniable signatures and can be seen as a simple realization of the later. More accurately, chameleon signatures represent a trade-off between simplicity and full functionality relative to undeniable signatures. While chameleon signatures are conceptually simpler and more efficient than the existing undeniable signature schemes, they also provide for a more restricted range of applications (due to their recipient-specific nature and the fact that they do not hide the recipient's identity). Yet, chameleon signatures provide a more practical alternative to undeniable signatures in many applications (such as for protecting the confidentiality of agreements as described above), as well as better analytical properties. One important characteristic of chameleon signatures is their non-interactive nature. We note that, as demonstrated by [Cha, JSI96], one can transform some of the traditional interactive undeniable signature schemes into undeniable signatures with non-interactive verification. This transformation uses idealized random oracle techniques (as in [FS86]) to remove interaction and requires the use of chameleon commitment schemes (but applied to the confirmation proof rather than to the signature generation as in our case). Still many of the advantages of chameleon signatures mentioned before – such as simplicity, efficiency and cryptographic assumptions – hold relative to the schemes of [Cha, JSI96]. On the other hand, the latter can be used in those applications covered by traditional undeniable signatures but not by chameleon signatures.

2 Overview of Chameleon Signatures

2.1 Chameleon Hashing

Here we provide a succinct overview of chameleon hashing, a tool that we use in an essential way in our signature schemes (see Section 3 for more details). We start by noting that chameleon hashing are basically non-interactive chameleon commitment schemes as introduced by Brassard, Chaum and Crepeau [BCC88]. Their use in our context, however, is not as a commitment protocol but as a hash function on which a regular signature function is applied. Viewing them as hash functions helps in understanding their role in the context of chameleon signatures, namely, as collision resistant functions with a trapdoor for finding collisions.

A *chameleon hash function* is associated with a pair of public and private keys (the latter called a *trapdoor*) and has the following properties.

1. Anyone who knows the public key can compute the associated hash function.
2. For those who don't know the trapdoor the function is collision resistant in the usual sense, namely, it is infeasible to find two inputs which are mapped to the same output.
3. However, the holder of the trapdoor information can easily find collisions for every given input.

The actual definition of chameleon hashing, presented in Section 3, also adds a requirement on the output distribution of these functions (which, in particular, need to be randomized). We note that the name "chameleon" (borrowed from [BCC88]) refers to the ability of the owner of the trapdoor information to change the input to the function to any value of his choice without changing the output.

Building on existing chameleon commitments we describe several constructions of chameleon hashing. We show schemes based on standard cryptographic assumptions, such as the hardness of factoring or computing discrete logarithms. The efficiency of these constructions is similar (or better) to that of regular digital signatures.

2.2 Chameleon Signatures

Why is chameleon hashing worth considering in our context? Consider first the standard practice of applying a regular digital signature (say RSA or DSS) to a collision resistant hashing of a given message (e.g., using the SHA algorithm). Now, replace

the standard hash function with a chameleon hash H_R where R (for Recipient) is a party holding the trapdoor information for H_R, and for whom the signature is intended. The newly obtained signature scheme has some interesting properties:

1. As in regular digital signatures, the signer S cannot repudiate (or deny) a signature he generated since he cannot find collisions in the hash.

2. The recipient cannot prove to any third party that a given signature of S corresponds to a certain message since R could "open" the signed message in any way he wants as he can find collision using the trapdoor information of the hash.[2]

3. Signatures are *recipient-specific*, namely, if the same message is intended for two different recipients then the signer needs to sign it twice, once for each recipient (since the chameleon hash functions are specific and different for each recipient).

In other words the signatures are at the same time non-repudiable (property 1) and non-transferable (property 2). Non-transferability means, that only the intended recipient can be convinced of the validity of a signature, while no third party can be convinced of that validity (even with the help of the recipient) or get any other information on the contents of the signed message. This is the core property that protects our signatures from the danger of uncontrolled dissemination. However, how can the non-repudiation property be implemented if no third party can determine the validity or invalidity of a signature?

The point is that, following the same principle of undeniable signatures, a signature as above can be validated or denied *in collaboration with the signer*. In case of a legal dispute between R and S, the latter can be summoned to appear before a judge who can request that S accept the signature as claimed by R or otherwise deny it. For denying a signature we show a very simple procedure which draws on the property that if R presents an invalid signature then S can show collisions in the chameleon hash function H_R. This will be sufficient proof of R 's cheating (as the function is otherwise collision resistant for S). On the other hand if R is honest there is no way for S to deny the signature. Furthermore, even if a judge (or other party) who got the validation (denial) of the signature from the signer, provides a third party (e.g., a journalist or competitor) with all the information he got from S, there is still no way for that third party to validate (deny) the signature.

We call the signatures obtained by following the above approach *chameleon signatures* (again the pictorial name refers to the ability of the recipient to "open" the signature contents in any desired way). There are additional technical issues to take care of (and we do that in the next sections) but the above description provides a good idea of the essence of chameleon signatures.

The combination of regular digital signature schemes and chameleon hashing results in simple and efficient constructions of chameleon signatures. The total cost of such schemes is about twice the cost of regular digital signatures (e.g. RSA or DSS). The security of our chameleon signatures is proven based on standard cryptographic assumptions. In particular, we prove the unforgeability property solely on the unforgeability of the underlying digital signature in use. The non-repudiation property is derived from the same assumptions needed to build chameleon hash functions, e.g., the hardness of factoring or computing discrete logarithms. The non-transferability property depends also on the underlying chameleon hash function. Remarkably, we can show constructions of chameleon signatures where non-transferability is achieved *unconditionally*, namely, the signed message is information theoretically hidden by the signature string.

3 Chameleon Hashing

Here we present a full definition of chameleon hashing and briefly describe some implementations using known chameleon commitment schemes.

A chameleon hash function is associated with a user R who has published a public (hashing) key, denoted HK_R, and holds the corresponding secret key (the trapdoor for finding collisions), denoted CK_R. The pair of public and secret keys is generated by R according to a given generation algorithm. The public key HK_R defines a *chameleon hash* function, denoted $\text{CHAM-HASH}_R(\cdot,\cdot)$, which can be computed efficiently given the value of HK_R. On input a message m and a random string r, this function generates a hash value $\text{CHAM-HASH}_R(m,r)$ which satisfies the following properties:

Collision resistance There is no efficient algo-

[2] In this sense the signature is like an appended message-independent signature (e.g., a hand-written signature) that can be "cut-and-pasted" by R from one document to another.

rithm that on input the public key HK_R can find pairs m_1, r_1 and m_2, r_2 where $m_1 \neq m_2$ such that $\text{CHAM-HASH}_R(m_1, r_1) = \text{CHAM-HASH}_R(m_2, r_2)$, except with negligible probability.

Trapdoor collisions There is an efficient algorithm that on input the secret key CK_R, any pair m_1, r_1, and any additional message m_2, finds a value r_2 such that $\text{CHAM-HASH}_R(m_1, r_1) = \text{CHAM-HASH}_R(m_2, r_2)$.

Uniformity All messages m induce the same probability distribution on $\text{CHAM-HASH}_R(m, r)$ for r chosen uniformly at random. (In particular, from seeing $\text{CHAM-HASH}_R(m, r)$ for randomly chosen r nothing is learned about the message m.) This condition can be relaxed to require that the above distributions are not necessarily identical for all messages but computationally indistinguishable [GM84].

In the above definition we do not specify the exact notions of efficiency and of negligible probability. These can be modeled by polynomial bounds or be quantified by explicit (concrete) time and probability bounds. We note that the probability in finding collisions (in the first condition) depends on the internal random bits of the collision-finder algorithm as well as on the random choices of the algorithm that generates the pair of private and public keys for the hash (e.g., there may be such pairs where finding collisions is easy but the generation algorithm will output them with only negligible probability).

Chameleon hash functions are intended to act on arbitrarily long messages and generate an output of fixed (or bounded) length. An important property of chameleon hashing is presented in the next lemma, and is easy to verify.

Lemma 1 *The composition of a chameleon hash function and a (regular) collision-resistant hash function (where the latter is applied first) results in a chameleon hash function.*

Thus, if we have a collision-resistant hash function that maps arbitrary messages to hash values of length ℓ, e.g. $\ell = 160$ for SHA-1 [fST95], then it is enough to design a chameleon hash function that hashes elements of length ℓ. We use this fact in our implementations below as well as in the applications of these functions to chameleon signatures. In some cases, even if the chameleon hash function that we construct directly supports arbitrary length messages, it will be more efficient to first apply a (faster) regular collision-resistant function to the message and then the chameleon hash.

Chameleon commitments. Chameleon hashing is rooted in the notion of *chameleon commitment* (also called chameleon blobs or trapdoor commitments) which were first introduced by Brassard, Chaum and Crepeau [BCC88] in the context of zero-knowledge proofs. Any chameleon commitment scheme with a non-interactive commitment phase induces a chameleon hash function, and vice versa. To see this notice that the collision-resistant property of chameleon hashing implies that the function $\text{CHAM-HASH}_R(m, r)$ would bind a committer to a certain value m as he cannot open the commitment in two different ways. The trapdoor property gives the "chameleon" property as it enables R (the recipient of the commitment in this case) to open the hash string $hash$ to any possible pre-image value m'. The uniformity property prevents a third party, examining the value $hash$, from deducing any information about the hashed message.

Implementations. Being a central tool in the construction of chameleon signatures it is important to show efficient constructions of chameleon hashing based on standard cryptographic assumptions. We describe several constructions of chameleon hash functions based on existing chameleon commitment schemes. In particular, we show an efficient construction based on the hardness of computing discrete logarithms (next subsection), and one based on factoring (see Appendix A). A scheme based on claw-free permuations will appear in the full paper.

Chameleon Hashing Based on Discrete Log

Chameleon Hashing based on discrete log appears in Figure 1. This solution for chameleon hashing is based on a well known chameleon commitment scheme due to Boyar et al. [BKK90] (see also [BCC88]).

The collision resistance property of the scheme in Figure 1 (for anyone that does not know x) is based on the hardness of computing discrete logarithms. The knowledge of x, the trapdoor information, enables computing trapdoor collisions, namely, for any given m, m' and r all in Z_q^* a value $r' \in Z_q^*$ can be found such that $\text{CHAM-HASH}_y(m, r) = \text{CHAM-HASH}_y(m', r')$. This is done by solving for r' in the equation $m + xr = m' + xr' \bmod q$. From this we can also see that the uniformity property of CHAM-HASH also holds.

Setup:	Prime numbers p and q such that $p = kq + 1$, where q is a large enough prime factor
	An element g of order q in Z_p^*
	The private key CK_R is $x \in Z_q^*$
	The public key HK_R is $y = g^x \bmod p$ (p, q, g are implicit parts of the public key)
The function:	Given a message $m \in Z_q^*$ choose a random value $r \in Z_q^*$
	define the hash as $\text{CHAM-HASH}_y(m, r) = g^m y^r \bmod p$

Figure 1. Chameleon Hashing – based on Discrete Log

4 The Basics of Chameleon Signature Schemes

Here we present in some detail the basic components and requirements of a chameleon signature scheme. As we have stated previously a chameleon signature is generated by digitally signing a chameleon hash value of the message. In Section 4.1 we introduce the basic functions associated with a chameleon signature scheme. Then in Section 4.2 we discuss the limitations of the basic scheme and motivate the more involved details of our complete solutions which are presented in Section 5.

4.1 The basic components

We start by describing the setting for Chameleon Signatures. The setting defines the players and the agreed upon functions and keys.

Players: Signer S and recipient R. In addition, we shall refer to a judge J who represents a party in charge of settling disputes between S and R, and with whom S is assumed to collaborate.

Functions: The players agree on:

- A digital signature scheme (e.g., RSA, DSS) which defines a set of public and private keys associated with the signer, and the usual operations of signing, denoted by SIGN, and verification, denoted by VERIFY. That is, SIGN takes as input a message m and returns the signature on the message under the signer's private key, and VERIFY takes a message and its signature and uses the signer's public key to decide on the validity (or invalidity) of the signature. We assume this signature scheme to be unforgeable [GMR88]. (In practice, this usually requires an appropriate encoding of the signed information, e.g. using a cryptographic hash function.)
- A chameleon hashing function which defines a set of public and private keys associated with the "owner" of the hash, and the operation CHAM-HASH for generating a hash on a message. In our setting the "owner" of the hash function will be the recipient.

Keys:

- The signer S has a public and private signature keys which correspond to the agreed upon signature scheme, denoted by VK_S and SK_S, respectively.
- The recipient R has a public and private keys as required by the agreed upon chameleon hashing scheme. These are denoted by HK_R and CK_R, respectively.

We can assume that all public keys are registered with some trusted certification authority (depending on the legal requirements of a given application). It must be noted that when a person registers the public data required for the chameleon hash he must prove that he knows the trapdoor information (i.e. the corresponding private key) for the hash[3].

We now present the three basic stages of a chameleon signature scheme and their basic implementation (more complete details are given in subsequent sections).

4.1.1 Chameleon Signing Given a message m, and keys SK_S, and HK_R, the signer generates a signature for m in the following manner: The signer chooses a random element r and computes $hash = \text{CHAM-HASH}_R(m, r)$ and $sig = \text{SIGN}_S(hash)$. The triple $SIG(m) = (m, r, sig)$ is then transmitted from S to R.

Note: We stress that it is important in order to

[3] Proving knowledge of the secret key by the registrant R is required to avoid the case in which the private key is chosen by or known only to a third party P. In such a case, P will get convinced of signatures signed for R since only he knows the trapdoor information and not R. For a discussion on this topic see [JSI96].

guarantee non-transferability (see Section 4.3) that the values m and r transmitted from S to R are *not* part of the information signed under the function SIGN. The channel between S and R can still be authenticated as long as that authentication is repudiable – e.g. using a symmetric key MAC scheme (see also [DNS98]).

4.1.2 Chameleon Verification Given as input the triple (m, r, sig), and the public keys of both signer (VK_S) and recipient (HK_R) a chameleon verification is performed as follows. The value $hash$ is computed as $\text{CHAM-HASH}_R(m, r)$ and the string sig is verified using the VERIFY function of the standard signature scheme under VK_S (i.e., whether sig is the valid signature of S on $hash$). The chameleon verification denoted by CHAM-VER outputs that the signature is a *proper* (resp., *improper*) chameleon signature when the signature verification outputs valid (resp., invalid).
Note: This verification function is sufficient for R to get assurance of the validity of S's signature (i.e., R is assured that S will not be able to later deny the signature). However, for any other party, a successful verification represents no proof that S signed a particular message since R (knowing CK_R) could have produced it by himself.
Terminology: We will use the notation $\text{CHAM-VER}_{R,S}(m, r, sig)$ to denote that the chameleon verification is applied to the triple (m, r, sig) using the public keys of R and S. If the output of this function is "proper" we call (m, r, sig) an (R, S)-*proper* triple. (We omit the pair (R, S) when these values are clear from the context.)

4.1.3 Dispute In case of a dispute on the validity of a signature, R can turn to an authorized judge J. The judge gets from R a triple $SIG(\widehat{m}) = (\widehat{m}, \widehat{r}, \widehat{sig})$ on which J applies the above CHAM-VER function. This first test, by the judge, is to verify that the triple is an (R, S)-proper signature on the message $\widehat{m}$. If this verification fails then the alleged signature is rejected by J. Otherwise, J summons the signer to deny/accept the claim. In this case we assume the signer cooperates with the judge (say, by force of law). J sends to S the triple $SIG(\widehat{m})$. If the signer wants to accept the signature he simply confirms to the judge this fact. On the other hand, if S wants to claim that the signature is invalid he will need to provide a collision in the hash function, i.e. a value $m' \neq \widehat{m}$, and a value r' such that $\text{CHAM-HASH}_R(m', r') = \text{CHAM-HASH}_R(\widehat{m}, \widehat{r})$. Notice that S can always present such a pair m', r' if the signature $SIG(\widehat{m})$ is invalid (since in this case $\widehat{sig}$ was originally generated by S with some pair m, r different than $\widehat{m}, \widehat{r}$). In other words, by claiming a false signature, the recipient R provides S with a collision in the function CHAM-HASH_R. Yet, if the signature $SIG(\widehat{m})$ is valid then S cannot find collisions and the signature cannot be repudiated. Hence, the validity of $SIG(\widehat{m})$ is rejected by J if S can present collisions to the hash function, and accepted otherwise.
Remark. Note that in the dispute protocol when a signer decides to accept a signature $SIG(m')$ then no "proof" is provided to J of this validity (S only declares acceptance). As a consequence, it could be the case that the signature is not valid but S decides to accept it now and maybe to deny it at a later time. In applications where this situation is considered a real concern it can be overcome by running the denial procedure in two stages: first, J sends to S the alleged message m' and signature sig. If S accepts the signature as valid then it needs to send to J a value r that J verifies to be equal to r'. If he claims the signature to be invalid then the denial procedure is completed as specified above (that is, J sends r' to S and S responds with a collision).

4.2 Enhancements to the Basic Scheme

The above scheme conveys the main idea of our constructions but suffers from several limitations which we need to solve in order to obtain a complete and practical chameleon signature scheme.

The recipient's identity. In the above scheme the value $hash$ is taken to be a number without specification how it was generated or from what range it was computed. Based on this the following attack can be mounted by either the signer or the recipient.

The signer (recipient) will simply claim that the hash was generated under a different hash function, and give the specific hash function which would satisfy the required value. If this attack is activated by the signer he could generate a "dummy" recipient and claim that the signature was generated for this party, which means that the signer could completely disavow the signature. In the case where the recipient is the attacker the effect of the attack is that it will disable the signer from proving that a claimed signature is a forgery. In order to prove the forgery the signer needs to provide a collision in the hash function which is done by using the original signed value and the new claimed value of the signature. When both these values were computed under the

same hash they are themselves a collision in the hash. But now if the recipient changes the hash function, then the signer does not have a collision.

Thus, in order to disable these attacks the specific hash function used in the computation must be added under the signature. Notice that it isn't enough to include the information about the hash within the document being signed, as this can be changed when the whole document is being forged. Thus, we need to bind the hash function to the hashed value by signing (with S's signature) both *hash* and the specifics of the hash, i.e. HK_R(and possibly other information if desired).

This is the reason why the identity of R is not concealed in our schemes. As the recipient can prove that he is the owner of the hash by showing possession of the private key. For applications in which the disclosure of R's identity is to be avoided this can be achieved by combining our results with the techniques of [GKR97b].

Exposure-freeness. Notice that if a judge J summons a signer S to deny a forgery $SIG(\widehat{m}) = (\widehat{m}, \widehat{r}, \widehat{sig})$ then it must be that $\widehat{sig}$ has been generated by S as the signature on some message m using a string r for which $\text{CHAM-HASH}_R(m, r) = \text{CHAM-HASH}_R(\widehat{m}, \widehat{r})$. (We are guaranteed of that since J first verifies $SIG(\widehat{m})$ under the chameleon verification procedure.) If R's claim is false, S can always show a collision in the hash function by exposing the real m and r originally used for that signature. However by doing so S is disclosing information (i.e., the existence of the signature on m). This may be undesirable in certain applications. We will require that the signer will be able to deny a forgery without exposing the real contents of any message which he signed, and will make this a property of all our chameleon signature schemes. (The solutions we present achieve this goal in a strong sense, namely, S can deny the signature by presenting a random message m' totally unrelated to the original value m or to any other message signed by S.) We shall refer to this property as *exposure freeness*. We present a specific example on how to achieve exposure freeness (Section 5).

Memory requirements. As observed above, the signer needs to participate in a denial of a signature only if the $\widehat{sig}$ component of an alleged signature $SIG(\widehat{m})$ corresponds to a signature generated by him (for R) for some message m. If this signature is in fact a forgery, in order to deny it in our schemes, the signer will need to find out what was the real message corresponding to $\widehat{sig}$. One solution is that the signer will store all his signatures together with the corresponding signed message, as there is no means of computation by which he can extract the original message out of the hash. While this may be practical in some applications it may be prohibitive in others. We show how to relax this need by including the storage of this information in the signature string held by the recipient, thus this information is implicitly stored by R . (Note that this is a reasonable approach as R must always store the signatures and corresponding messages for the case of eventual disputes.) This is done in the following manner, the signer will have some private key, k, under which he encrypts both m and r generating $enc_k(m, r)$. This value is signed together with *hash* and the identity of R. (We note that it suffices to encrypt a digest of m computed under a collision-resistant hash function rather than the entire message m. In this case S signs by applying CHAM-HASH to this digest of m rather than to m itself.) The encryption must be semantically secure [GM84] and can be implemented using a symmetric or asymmetric cryptosystem. Note, that if this option is used then the non-transferability property discussed below cannot be achieved information-theoretically.

4.3 Security Requirements

Here we summarize the security properties that we require from a chameleon signature scheme. Formal definitions will be presented in the final version of this paper.

We shall say that a signature scheme carried out by a signer S, a recipient R and a judge J, which is composed of the functions described in Section 4 is a *secure chameleon signature* scheme if the following properties are satisfied. (In what follows we refer as a *third party* to any party different than the signer and recipient.)

Unforgeability. No third party can produce an (R, S)-proper triple $SIG(m) = (m, r, sig)$ not previously generated by the signer S. The intended recipient R can produce an (R, S)-proper triple (m, r, sig) only for values sig previously generated by S.

Non-transferability. Except for the signer himself, no one can prove to another party that the signer produced a given triple $SIG(m) = (m, r, sig)$ for any such triple. This should be true for the recipient and for any third party (including one – say a judge – that participated in a denial/confirmation protocol with the signer).

Denial. In case of dispute, if the signer S is presented with a triple $SIG(m) = (m, r, sig)$ not pro-

Chameleon Signature Generation - CHAM-SIG

Input of Signer: Message $m \in Z_q^*$
private signing key of S, SK_S
HK_R of R, i.e. $HK_R = y(= g^x \bmod p), g, p, q$

1. Generate the chameleon hash of m by choosing a random $r \in Z_q^*$ and computing $hash = \text{CHAM-HASH}_R(m, r) = g^m y^r \bmod p$
2. Set $sig = \text{SIGN}_S(hash, HK_R)$
3. The signature on the message m consists of $SIG(m) = (m, r, sig)$.

Chameleon Signature Verification - CHAM-VER

Input: $SIG(m) = (m, r, sig)$
public verification key of S, VK_S
HK_R of R, i.e. $HK_R = y, g, p, q$

1. Compute $hash = \text{CHAM-HASH}_R(m, r) = g^m y^r \bmod p$
2. output $= \begin{cases} \text{proper} & \text{VERIFY}_S((hash, HK_R), sig) = \text{valid} \\ \text{improper} & otherwise \end{cases}$

Figure 2. Dlog-based Chameleon Signatures – Generation and Verification

duced by him, then S can convince a (honest) judge J to reject $SIG(m)$.

Non-repudiation. In case of dispute, if the signer S is presented with a triple $SIG(m) = (m, r, sig)$ produced by him, then S cannot convince a (honest) judge J to reject $SIG(m)$.

Exposure freeness. A chameleon signature scheme is exposure free if the signer can deny a false signature (i.e., a triple (m, r, sig) not produced by him) without exposing any other message actually signed by him.

5 A Full Chameleon Signature Scheme

In this section we shall describe, for concreteness, a specific system for Chameleon Signatures which fully satisfies the above functionality and security requirements. The implementation described below achieves the property of being *exposure-free*, i.e. in case of denial the signer will be able to prove the invalidity of the signature without exposing any of his signed messages. We omit details of memory management, namely, whether the message m and its signature are kept by the signer for possible disputes or whether an encryption of the hashed message is added to the signature. The techniques for solving these additional issues, described in Section 4.2, are independent of the type of chameleon hash used and can be easily incorporated here.

The Chameleon Hashing which will be employed in this specific solution is the discrete log based chameleon hashing described in Section 3. In addition to the CHAM-HASH function we will use the functions SIGN and VERIFY as defined by some specific signature scheme (e.g., DSS or RSA with the appropriate encoding of the signature arguments as discussed in Section 4.1), and for which S generates a pair of private and public keys.

Based on the above we can define the function CHAM-SIG for chameleon signature generation, and the function CHAM-VER for chameleon verification, these are described in Figure 2. The construction follows the basic scheme of Section 4.1 with the incorporation of the specifics of the Chameleon Hash under the signature. We remark that the message m used as input to the signature can be first hashed using a collision-resistant hash function (e.g., SHA-1) and the result of this hashing used as the input to the chameleon hash function (see Lemma 1).[4]

As was described in Section 4.1 in case of dispute the signer will be presented with a triple $SIG(m') = (m', r', sig)$ which passes the CHAM-VER verifica-

[4] Typically, this will result in several layers of hashing. First a collision-resistant hashing of the message is computed; then, the chameleon hashing is applied to this value; finally, the resultant chameleon hash value and other items to be signed are input to some standard encoding procedure (usually based on hash functions as well) for the specific digital signature scheme in use.

Generate Collision

Input: a forgery $SIG(m') = (m', r', sig)$

1. S retrieves the original values m, r used to compute sig.
   It holds that $g^m y^r \bmod p = g^{m'} y^{r'} \bmod p$, while $m \neq m'$ [a].
2. S computes $x = \frac{m-m'}{r'-r} \bmod q$.
3. S chooses any message $\bar{m} \in Z_p^*$ and computes $\bar{r} = \frac{m+xr-\bar{m}}{x} \bmod q$
4. Output $(\bar{m}, \bar{r})$.

[a] The retrieval of m and r can be done out of S's archives or using the encryption technique described in Section 4.2.

Figure 3. Generating Collisions – Discrete Log based

tion, i.e. sig is a possible signature generated by the signer for the message m'. The signature will be considered invalid if the signer can provide a collision in the hash function. Furthermore, disavowing the signature should be achieved without exposing the original message which the signer signed. In our discrete-log based solution, once the recipient has presented a forgery (which passes the chameleon verification) then not only will the signer be able to disavow the specific signature but he will also be able to expose R 's private key x. Knowledge of x, in turn, enables the signer to disavow all other messages which he signed using this chameleon hashing, since by knowing the trapdoor one can find arbitrary collisions. The protocol carried out by the signer for extracting the secret key and producing a random collision in the hash is described in Figure 3.

Theorem 1 *Assuming a secure digital signature scheme and the hardness of computing discrete logs, the above procedures form a chameleon signature scheme satisfying the properties of non-transferability, unforgeability, non-repudiation, denial, and exposure freeness.*

Proof.

Non-transferability. Given a signature $SIG = (m, r, sig)$ generated by the signer S for a recipient R, the recipient cannot convince a third party of its validity. This is due to the fact that for every possible message m', R can compute a value $r' \stackrel{\text{def}}{=} \frac{m+xr-m'}{x} \bmod q$ such that $\text{CHAM-HASH}_R(m, r) = \text{CHAM-HASH}_R(m', r')$ (Section 3). Thus, (m', r', sig) is an (R, S)-proper signature. Furthermore, since for every possible message m' there exists exactly one value r' that produces a proper triple (m', r', sig) then nothing is learned about the value of m from seeing the signature string sig. Thus non-transferability is achieved unconditionally, i.e. in the information theoretic sense. In addition, no third party can prove the validity of the signature as it may be assumed that he is in collusion with the recipient.

Unforgeability. No third party can generate an (R, S)-proper triple $SIG(m) = (m, r, sig)$ which has not been previously generated by the signer S, as this requires either to break the underlying digital signature scheme, or to find collision in the chameleon hash function which, in turn, implies computing the secret trapdoor information of R . The recipient cannot generate an (R, S)-proper triple $SIG(m) = (m, r, sig)$ with a component sig which has not been previously signed by the signer as this requires to break the underlying digital signature scheme.

Non-repudiation. Given an (R, S)-proper triple $SIG(m) = (m, r, sig)$ generated by the signer, under the specified hash key, g, y, p the signer cannot generate another (R, S)-proper triple $SIG(m') = (m', r', sig)$ for $m \neq m'$ as this would be equivalent to computing the secret trapdoor information x, which we assume to be infeasible by the hardness of the discrete log problem.

Exposure freeness. Since we assume the underlying digital signature to be unforgeable, the signer S may be required to deny a signature only for a triple $SIG(m') = (m', r', sig)$ which is (R, S)-proper but not originally generated by S. In this case, S must possess another proper triple $SIG(m) = (m, r, sig)$ that he really signed. Using these values S extracts the secret trapdoor information as described in Figure 3 . Given this trapdoor the signer can deny the signature

by presenting a collision using any message of his choice or by presenting the trapdoor itself. Clearly, the exposure of the recipients trapdoor to the signer should serve as an additional deterrent for the recipient from producing forgeries.

6 Convertibility

The notion of convertibility of undeniable signatures was introduced by Boyar, Chaum, Damgard and Pedersen [BCDP90]. The idea is that an undeniable signature will be transformed into a regular publicly verifiable (non-repudiable) signature by publishing some information. There are also variations to the notion of convertibility, i.e. complete and selective. Complete convertibility transforms all the signatures generated under the same key, while selective convertibility transforms only a single signature. Secure solutions to the problem of convertible undeniable signatures appear in Damgard and Pedersen [DP96] and Gennaro, Krawczyk, and Rabin [GKR97a].

In Section 4.2 we introduced a method in order to circumvent the need for the signer to store the message. Using this same technique we can enable convertibility. The mechanism was that the signer includes under his signature an encryption of the signed message[5]. To enable convertibility the signer needs to encrypt the message using a public key encryption. Furthermore, the signer will commit to the encryption public key that he is using by signing this public key together with the encryption of the message (this prevents problems similar to those pointed out in [AN95] when signing an encryption). Thus, the signer is committed to some encryption string. However, the contents of the encrypted string cannot be learned by any third party due to the semantic security of the encryption. Now, we can achieve selective convertibility by having the signer expose the random bits used for the specific probabilistic encryption of the signed message, and complete convertibility by exposing the decryption key. (This commitment is unique given the one-to-one nature of the encryption function.)

Acknowledgments

We would like to thank Mihir Bellare, Ivan Damgard, Shimon Even and Daniele Micciancio for helpful discussions.

[5] In a sense, this solution can be seen as dual to the solution in [DP96] that is based on the idea of encrypting a signature while here we sign an encryption.

References

[AN95] R. Anderson and R. Needham. Robustness principles for public key protocols. In *Crypto '95*, pages 236–247, 1995. LNCS No. 963.

[BCC88] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *JCSS*, 37(2):156–189, 1988.

[BCDP90] J. Boyar, D. Chaum, I. Damg$R$ard, and T. Pedersen. Convertible undeniable signatures. In *Crypto '90*, pages 189–205, 1990. LNCS No. 537.

[BKK90] J. F. Boyar, S. A. Kurtz, and M. W. Krentel. A discrete logarithm implementation of perfect zero-knowledge blobs. *Journal of Cryptology*, 2(2):63–76, 1990.

[CA89] David Chaum and Hans Van Antwerpen. Undeniable signatures. In *Crypto '89*, pages 212–217, 1989. LNCS No. 435.

[Cha] David Chaum. Private Signatures and Proof Systems. US Patent 5,493,614 issued 02/20/1996.

[Cha90] D. Chaum. Zero–knowledge undeniable signatures. In *Eurocrypt '90*, pages 458–464, 1990. LNCS No. 473.

[Cha94] David Chaum. Designated confirmer signatures. In *Eurocrypt '94*, pages 86–91, 1994. LNCS No. 950.

[CvHP91] D. Chaum, E. van Heijst, and B. Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In *Crypto '91*, pages 470–484, 1991. LNCS No. 576.

[Dam87] I. Damgard. Collision free hash functions. In *Eurocrypt '87*, pages 203–216, 1987. LNCS No. 304.

[DNS98] C. Dwork, M. Naor, and A. Sahai. Concurrent zero-knowledge. In *Proc.* 30*th STOC*, pages 409–418. ACM, 1998.

[DP96] I. Damgard and T. Pedersen. New convertible undeniable signature schemes. In *Eurocrypt '96*, pages 372–386, 1996. LNCS No. 1070.

[DY91] Y Desmedt and M. Yung. Weaknesses of undeniable signature schemes. In *Crypto '91*, pages 205–220, 1991. LNCS No. 576.

[FOO91] A. Fujioka, T. Okamoto, and K. Ohta. Interactive bi-proof systems and undeniable signature schemes. In *Eurocrypt '91*, pages 243–256, 1991. LNCS No. 547.

[FS86] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. In *Crypto '86*, pages 186–194, 1986. LNCS No. 263.

[fST95] National Institute for Standards and Technology. Secure Hash Standard, April 17 1995.

[GKR97a] R. Gennaro, H. Krawczyk, and T. Rabin. RSA-based Undeniable Signatures. In *Crypto '97*, pages 132–149, 1997. LNCS No. 1294.

[GKR97b] R. Gennaro, H. Krawczyk, and T. Rabin. Undeniable Certificates. Manuscript, 1997.

[GM84] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28(2):270–299, April 1984.

[GMR88] Shafi Goldwasser, Silvio Micali, and Ronald L. Rivest. A digital signature scheme secure against adaptive chosen-message attacks. *SIAM J. Computing*, 17(2):281–308, April 1988.

[JSI96] M. Jakobsson, K. Sako, and R. Impagliazzo. Designated verifier proofs and their applications. In *Eurocrypt '96*, pages 143–154, 1996. LNCS No. 1070.

[JY96] M. Jakobsson and M. Yung. Proving without knowing: On oblivious, agnostic and blindfolded provers. In *Crypto '96*, pages 201–215, 1996. LNCS No. 1109.

[Mic96] M. Michels. Breaking and repairing a convertible undeniable signature scheme. In *ACM Conference on Computer and Communications Security*, 1996.

[Oka94] Tatsuaki Okamoto. Designated confirmer signatures and public-key encryption are equivalent. In *Crypto '94*, pages 61–74, 1994. LNCS No. 839.

[Ped91] T. Pedersen. Distributed provers with applications to undeniable signatures. In *Eurocrypt '91*, pages 221–242, 1991. LNCS No. 547.

[Rab78] M. Rabin. Digitalized Signatures. In R. Demillo and et.al, editors, *Foundations of Secure Computations*, pages 155–165. Academic Press, 1978.

A Chameleon Hashing Based on Factoring

We present an implementation of Chameleon Hashing based on factoring. It is a specific an efficient implementation of a general construction based on claw-free permutations which was introduced by Goldwasser, Micali and Rivest [GMR88] for building regular digital signatures, and used by Damgard [Dam87] to build collision resistant hash functions.

Choose primes $p \equiv 3 \bmod 8$ and $q \equiv 7 \bmod 8$ and compute $n = pq$.

Input: Message $m = m[1]...m[k]$
$HK_R = n$ as defined above
Choose random value $r \in Z_n^*$;
$hash := r^2 \bmod n$
for $i = 1$ to k
$hash := 4^{m[i]} hash^2 \bmod n$

Figure 4. Chameleon Hashing – Based on Factoring

Computation analysis. The number of operations required to compute this chameleon hash is $|m|$ squarings and up to $|m|$ multiplications (by 4) mod n. Typically, for our application, m is a hashed message and then the expected number of multiplications will be $|m|/2$, and $|m|$ itself about 160 bit-long only. Thus the total cost is significantly lower than a full long exponentiation mod n and, in particular, than an RSA signature. Note, that the above function (if we take $m[1]$ to be the most significant bit of m and $m[k]$ the least significant) computes to $\text{CHAM-HASH}(m, r^2) = 4^m (r^2)^{2^{|m|}}$.