

Sanitizable Signatures

Giuseppe Ateniese¹, Daniel H. Chou¹, Breno de Medeiros², and Gene Tsudik³

¹ Johns Hopkins Univ.; Dept. of Comp. Sci.; 3400 N. Charles Street; Baltimore, MD 21218, USA. ateniese@cs.jhu.edu, dchou@cs.jhu.edu

² Florida State Univ.; Dept. of Comp. Sci.; Tallahassee, FL 32306, USA. breno@cs.fsu.edu

³ Univ. of California; D. Bren Sch. of Inform. and Comp. Sci.; Dept. of Comp. Sci.; Irvine, CA 92697, USA. gts@ics.uci.edu

Abstract. We introduce the notion of *sanitizable signatures* that offer many attractive security features for certain current and emerging applications. A sanitizable signature allows authorized semi-trusted censors to modify – in a limited and controlled fashion – parts of a signed message without interacting with the original signer. We present constructions for this new primitive, based on standard signature schemes and secure under common cryptographic assumptions. We also provide experimental measurements for the implementation of a sanitizable signature scheme and demonstrate its practicality.

1 Introduction and Motivation

In government, military and corporate environments, information is often compartmentalized in a way that one’s role or security clearance determines access rights with respect to a resource, such as a database or a document. Thus, two subjects with different security clearances can “see” the same information with varying granularity of detail. For example, the United States Government sometimes releases certain previously classified documents in “sanitized” form, often as a result of a request made through the Freedom of Information Act (FOIA). A document thus released is usually sprinkled with blacked-out sections which, for various reasons, remain confidential. More specifically, individual words, sentences, paragraphs and even entire sections of a document can be either deleted or substituted with dummy data prior to being released.

Now, suppose that someone needs to refer to, or cite from, a sanitized document. In this case, to avoid liability, it is necessary to ascertain the source and the integrity of the document. Plain digital signatures (e.g., RSA or DSA) provide the means to achieve both source authentication and data integrity. More exotic constructs, such as *Redactable Signatures* [23], allow anyone to obtain a valid signature of the redacted document without any help from the original signer. However, there are situations where a duly authorized third party (censor) may need to *modify* the document in some controlled and limited fashion. In doing so, the authorized censor needs to somehow come up with a valid signature for the updated document, without contacting the original signer. There could be

many possible reasons for not asking the original signer to re-sign, including: (1) the signer’s key has expired, (2) the original signature was securely time-stamped via, e.g., [18], (3) the signer may not be reachable/available, (4) each new signature would cost too much, either in terms of real expense or in terms of computation. In this paper, we introduce the notion of *sanitizable signatures* precisely in order to address these needs.

Informally, a *Sanitizable Signature Scheme* allows a semi-trusted censor to modify *designated* portions of the document and produce a valid signature on the legitimately modified document without any help from the original signer. These designated portions of the document are blocks or segments explicitly indicated as mutable under prior agreement between the signer and the censor. The censor can produce a valid signature only if it modifies these portions and no other parts of the message.

To illustrate the utility of sanitizable signatures, the rest of this section discusses several potential application scenarios.

1.1 Multicast and Database Applications

Sanitizable signatures are quite well-suited for customizing authenticated multicast transmissions. For example, in a subscription-based internet multimedia database, sponsors may wish to insert personalized commercials into messages at various points of the broadcast. It is desirable to authenticate these messages to allow the subscribers to distinguish legitimate contents from spam. Since real-time authentication may be too costly, one solution is for each vendor to sign the commercial once and allow the database administrator to customize the individual commercials by replacing the generic identity field with the actual subscriber’s identity, at various points of the commercial. This way, the subscriber can verify that the commercial comes from a legitimate source (i.e., it is not spam) and the sponsors do not have to sign each customized broadcast. Furthermore, the database administrator is not forced to divulge personal information of its subscribers without their consent.

A related application of sanitized signatures is editing movie content. Depending on the age of the subscriber, the administrator can replace offensive language with watered-down substitutes rather than blip out the words. Again, sanitized signatures provides the desired benefits.

In the same vein, sanitizable signatures can be used in outsourced database applications. Database outsourcing [19] is a recent and important industry trend whereby a Database Service Provider offers adequate resources to host its clients’ databases as well as mechanisms to efficiently manipulate and access outsourced data. Database outsourcing poses numerous security challenges since it involves a client storing its data at an external – and often untrusted – provider site. To this end, it is essential to protect the integrity and authenticity of that data from both malicious outsider attacks and the Database Service Provider itself. This is usually achieved by having the client sign each database record before outsourcing [21]. Later, when a user queries a database owned by a client of the Database Service Provider but physically stored at the latter, the provider

acts as an authorized re-distributor of outsourced data. In this role, it needs to ensure that users – who obtain portions of the database (as replies to queries) – cannot redistribute the results and themselves become unauthorized “de facto” distributors. With the aid of sanitizable signatures, a query reply (i.e., a set of database records) can be manipulated by the provider in a way that each returned record is signed by its original owner (client), but personalized for the specific user who posed the query.

More generally, sanitizable signatures can be viewed as a valuable tool for combatting certain types of software piracy and unauthorized content distribution. If the actual content owner is off-line and an authorized on-line distributor is used to sell or supply content to users, the benefit of sanitizable signatures is the ability of the distributor to easily personalize signed (i.e., authentic) content for each user or each transaction. While this would clearly not put a stop to piracy (since multiple corrupt users can always trade ill-begotten content among themselves), it would preclude honest users from being duped by unauthorized or fraudulent re-sellers/re-distributors of valuable content.

1.2 Medical Applications

The additional functionalities and flexibility of sanitizable signatures may also help protect the privacy of medical records. Under the Health Insurance Portability and Accountability Act of 1996 (HIPAA), covered entities are required to comply with the Standards for Privacy of Individually Identifiable Health Information (the Privacy Rule) [38]. The Privacy Rule specifies the criteria for creating both de-identified and limited data sets from protected health information (PHI) for research purposes. In particular, covered entities must remove direct identifiers of the individual or of relatives, employers, or household members of the individuals before PHI can be legally released for research purposes.

Compare the scenarios of a cancer study and an epidemic study. The two studies require different temporal resolution when creating limited data sets from PHI. In a cancer study, the exact dates when treatments are administered to the patient may not be necessary to the study. It is important, however, to note the length of time between treatments. On the other hand, in an epidemic study, it may be necessary to include exact treatment dates so the limited data set could reveal trends and patterns necessary for creating an epidemiological model.

Sanitizable signatures can be used to ensure the integrity, authenticity, and anonymity of PHI in both cases. In general, sanitizable signatures can accommodate different level of data de-identification, supporting the “minimum necessary” disclosure standard of HIPAA Privacy Rule. This provides flexibility not available in redactable signatures.

1.3 Secure Routing

A crucial aspect of security in modern routing protocols is the protection of exchanges of connectivity information between routers. An important feature of

a major class of routing protocols – called *distance vector* – is the direct exchange of routing tables among neighboring routers. *Distance vector* protocols require each router to maintain tables where each entry contains a destination and a route metric (cost) to that destination. More advanced *path vector* protocols, in addition, require each router to maintain – for each routing table entry, i.e., for each destination – an actual shortest route/path to that destination. The best-known path vector protocol is the Border Gateway Protocol (BGP) [33] widely used in the Internet.

There have been several proposals for supporting authentication of origin and data integrity in routing protocols, typically via digital signatures (see, for example, [27] and [24]). Indeed, routing message authentication is imperative for resistance against powerful – especially, Byzantine – adversaries. While mounting Byzantine attacks against routing algorithms is generally difficult, the transitivity of trust implied by the very essence of distance and path vector algorithms compounds the impact of any successful attack. Protecting *link state* protocols against Byzantine attacks, as in [27], is simpler than the same task for distance or path vector protocols such as [24]. A general architecture for link state protocols with Byzantine robustness has been developed rather early on, in [31], whereas, no equivalent architecture for path vector protocols has been proposed.

The main challenge in authenticating path vector routing messages is that – unless we assume complete transitivity of trust – for each path vector, a separate signature by each hop in the route is required. The combined cost of verifying multiple per-hop signatures becomes a serious burden on intermediate routers. This can be mitigated by using *transitive signatures* [26, 6, 37], which allow anyone to use the public keys of routers to combine several edge signatures (where edges are a pair of adjacent routers along the route) into a single path signature (from the source or any intermediate router to the destination or a subsequent intermediate router).

Sanitized signatures provide an alternative mechanism. The main difference between using transitive and sanitized signatures is that the latter delegate the ability to aggregate signatures to specific routers, while transitive signatures allow any router to aggregate. The explicit delegation model afforded by sanitized signatures is more flexible, as it permits the implementation of arbitrary trust infrastructures with respect to route aggregation.

Finally, we observe that similar techniques are applicable in on-demand MANET routing protocols [10], such as Dynamic Source Routing (DSR) [22]. DSR uses flooding to discover a shortest path to a destination. A route is collected incrementally, during flooding propagation, with each router adding itself to the route as it processes a route request message. It is easy to see that sanitized signatures are also appropriate in this setting and offer the same benefits as in path vector protocols.

2 Related Work

Several concepts are related to sanitizable signatures, including incremental cryptography and homomorphic signatures, which encompass transitive, redactable and context-extraction signatures.

Incremental cryptography seeks to construct cryptographic primitives with an efficient update property. Namely, if an incremental cryptographic algorithm produces a value when applied to a document, then the value may be very efficiently re-computed on a variant of the document obtained by applying a pre-defined transformation rule – in particular, more efficiently than recomputing the algorithm from scratch with the new document. Incremental cryptography was defined in Bellare et al [3, 4], including applications to incremental hashing and signing. A separate construction of an incremental signature scheme with certain privacy properties has been provided by Bellare and Micciancio [5].

Incremental and sanitizable signatures are similar in that they support signature re-computation through a process different than initial signature generation; however, they differ in that the latter supports delegation of the ability to perform updates to another party, while the former provides a mechanism for the original signer to perform updates more efficiently than through re-signing an entire document.

Homomorphic signatures: In a series of talks, Rivest [34] proposed the design of signature schemes that allow “forgeries” of pre-determined types. More specifically, a signer would need his/her private key to generate a signature on a document, but arbitrary parties could use simply the knowledge of the public key to modify the document in locations and fashion pre-selected by the signer, and obtain a new signature on the transformed document without interaction with the original signer. This concept was then formalized as *homomorphic signature schemes* in [23]. A particular construction made possible through the use of homomorphic signature schemes is a redactable signature (also [23]). When a document is redacted, each redacted bit position is replaced with the same special symbol to represent the location of the deletions. Explicitly marking the locations of the redactions is necessary to thwart semantic attacks. A sanitized document can be view as a redacted document that allows arbitrary bit substitution in the location of the deletions. However, there are other fundamental differences between sanitizable signatures and redactable signatures. As with other homomorphic constructions, redactable signature schemes allow anyone with the knowledge of the public key to generate a valid signature on the redacted document. This property is not always desirable in a digital signature scheme. In contrast, only the censor would be able to generate a valid signature on a modified (sanitized) document. Moreover, in our basic construction, the signer can incontestably prove that the censor sanitized the document. Thus, sanitizable signatures provide (and require) greater accountability. Furthermore, once a signature is redacted, it is impossible to undo the redaction and recover the signature on the original message. On the other hand, the censor can undo

the changes to the mutable portions of the message and produce a “sanitized” signature that corresponds to the original message.

A related concept to redactable signature is that of content-extraction signatures [36]. These are essentially redactable XML signatures, where the redaction operation efficiently removes XML nodes – permitting customization of publishable information to comply with privacy and confidentiality demands of dynamic distributed applications.

Transitive signatures are essentially homomorphic signatures, where the operation in question is path concatenation on (undirected or directed) graphs. Optimized constructions for transitive signatures, more efficient than general homomorphic techniques, have been proposed [26, 6, 37]. The interest in transitive signatures stem from their potential applicability to secure routing in computer networks [12], by enabling route-path signature aggregation. Namely, if a secure routing protocol is implemented via router signatures on each hop, the computational load on routers does not scale well, as increasingly long chains of signatures need to be verified. Transitive signatures permit any intermediate routers to collapse routes to a single signed source-current router pair (or to contract the route in any other intended fashion), thus achieving better efficiency as well as security: In some cases it may not be in the interest of routers (specially edge routers) to disclose the topology of the (internal) network they protect.

We remark that sanitizable (as well as redactable) signatures can be employed to achieve the route-path reduction efficiently – but under different trust models. Transitive and redactable signatures require intermediate routers to know only the public key of previous routers in the path in order to remove their signature to the authenticated path. On the other hand, sanitizable signatures would permit routers to delegate the ability to remove their signature to specific trusted routers. We believe that this trust model is more flexible and more representative of practical security architectures, where only some entities are entrusted with security policies for a network, and allowed to “edit” or sanitize network-security related information on behalf of other entities, as discussed in section §1.3.

Automatic Sanitization of Internet Traffic: There exists an entire area of research on sanitizing raw Internet packet traces for sharing and research purposes. Most of this work studies different ways of anonymizing TCP/IP packet header fields, for instance see [32, 39]. The seminal work of Pang and Paxson [30] focuses on sanitizing also packet payloads and has been extended and generalized by Bishop et al. [8].

That line of research seeks to develop methods of expressing privacy policies and then to create tools that can interpret such policies to automatically sanitize Internet traffic [30, 8]. While not directly related to this paper (since they do not deal with cryptographic primitives, such as signatures), we believe that the techniques developed in [30, 8] could be combined with ours for mutual advantage.

3 Sanitizable Signatures

We define a sanitizable signature scheme as a secure digital signature scheme that allows a semi-trusted censor to modify certain *designated* portions of the message and produce a valid signature of the resulting (legitimately modified) message with no interaction with the original signer. More concretely, a sanitizable signature scheme must have the following properties:

1. *Immutability*. The censor should not be able to modify any part of the message that is not specifically designated as sanitizable by the original signer.
2. *Privacy*. Given a sanitized signed message with a valid signature, it is impossible for anyone (except the signer and the censor) to derive any information about the portions of the message that were sanitized by the censor. In other words, all sanitized information is unrecoverable.[†]
3. *Accountability*. In case of a dispute, the signer can prove to a trusted third party (e.g., court) that a certain message was sanitized by the censor.
4. *Transparency*. Given a signed message with a valid signature, no party – except the censor and the signer – should be able to correctly guess whether the message has been sanitized.

We further distinguish among two flavors of transparency: *weak* and *strong*. Weak transparency means that the verifier knows exactly which parts of the message are potentially sanitizable and, consequently, which parts are immutable.

In contrast, strong transparency guarantees that the verifier does not know which parts of the message are immutable and thus does not know which parts of a signed message could potentially be sanitizable.

Either transparency flavor can be beneficial depending on the specific application. We stress that strong transparency is not always better. In certain circumstances, *weak transparency* is actually preferable. For example, if a document originally signed by some government official is later released by a certain government agency – acting as a censor – under the Freedom of Information Act, the general public would likely prefer knowing which parts of the document could have been sanitized.

Our construction only provides for weak transparency. Accordingly, we only provide a formal security model for weak transparency, in terms of an indistinguishability property.

3.1 Model

In this section, a formal definition of a sanitizable signature is given in terms of the algorithms that constitute the scheme and their security properties.

A sanitizable signature scheme is a set of four efficient algorithms (as usual, efficiency is defined in terms of a security parameter):

[†] Unless of course the original message is stored by the signer and/or the censor.

Key generation: For simplicity, we assume that each party could potentially be a censor. Principal P_i uses this probabilistic algorithm to compute two public-private key pairs:

$$(pk_{sign}^i, sk_{sign}^i), (pk_{sanit}^i, sk_{sanit}^i) \xleftarrow{R} 1^k,$$

where k is a security parameter. The first set of keys is for a standard digital signature algorithm, while the second is useful to perform sanitization steps.

Sign: Takes as input a message m , a private signing key sk_{sign}^i , a public sanitization key pk_{sanit}^j , random coins r , and produces a signature

$$\sigma \leftarrow SIGN(m, r; sk_{sign}^i, pk_{sanit}^j).$$

Verify: A deterministic algorithm that, on input a message m , a possibly valid signature σ on m , a public signing key pk_{sign}^i and a sanitization key pk_{sanit}^j , outputs TRUE or FALSE:

$$VERIFY(m, \sigma; pk_{sign}^i, pk_{sanit}^j) \rightarrow \{TRUE, FALSE\}.$$

Sanitize: An algorithm that, on input a message m , a signature σ on m under public signing key pk_{sign}^i , a private sanitizing key sk_{sanit}^j , and a new message m' , produces a new signature σ' on m' .

$$\sigma' \leftarrow SANIT(m, \sigma, m'; pk_{sign}^i, sk_{sanit}^j).$$

We now discuss security requirements of this definition.

Security Requirements of Sanitizable Signatures A sanitizable signature as above should satisfy the following criteria:

Correctness. A signature produced by the SIGN algorithm should be accepted by the VERIFY algorithm:

$$\forall \sigma = SIGN(m, r; sk_{sign}^i, pk_{sanit}^j); \\ VERIFY(m, \sigma; pk_{sign}^i, pk_{sanit}^j) = TRUE$$

Unforgeability. Without the knowledge of the private signing key it is difficult to produce a valid signature on a message that verifies against the associated public key, *except by resorting to the sanitization process*. The exact formulation of this concept can be provided within an adversarial-game framework, detailed in 3.1.

Indistinguishability. It is the property that, for any pair of messages m_1, m_2 , and any choices of private signing key sk_{sign}^i , and public sanitizing key pk_{sanit}^j , the following distributions \mathcal{S}_1 and \mathcal{S}_2 are computationally indistinguishable:

$$\mathcal{S}_1 = \{\sigma; \sigma = \text{SIGN}(m_1, r; sk_{sign}^i, pk_{sanit}^j)\}$$

and

$$\mathcal{S}_2 = \{\sigma; \sigma = \text{SIGN}(m_2, r; sk_{sign}^i, pk_{sanit}^j)\},$$

where r is chosen uniformly at random in the coin space of the SIGN algorithm.

Identical Distribution. Values produced by the SANIT algorithm are distributed identically to those produced by the SIGN algorithm. In particular, if a signature σ on message m (with random coins r) is sanitized to signature σ' on message m' , then there exist coins r' for which σ' is an original signature on m' :

$$\text{SANIT}(m, \sigma, m'; pk_{sign}^i, sk_{sanit}^j) = \sigma' = \text{SIGN}(m', r'; sk_{sign}^i, pk_{sanit}^j).$$

The above formulation of a sanitizable signature is not the only reasonable one. For instance, the requirement that the sanitization algorithm produces the exact outputs as the sign algorithm is not necessary as long as its outputs are 1) accepted by the verification algorithm, and 2) indistinguishable from the outputs of the sign algorithm. We adopt the stricter formulation instead as it is still general enough to capture the constructions we propose; because it has the benefit of being easier to formulate and understand; and because of closer parallel with related research literature – see, for instance, the formulation of transitive signatures in [6].

Referring back to the more informal requirements at the beginning of this section, we point out that the indistinguishability requirement provides for privacy, while the identical distribution implies the weak transparency property.

The unforgeability requirement (typical of signature schemes) involves some subtleties in the case of sanitizable signatures, as the sanitization process is a bona-fide forgery algorithm. In order to formulate this concept more precisely it is necessary to consider a stateful signer, since one must keep track of all previously issued signatures and queries to the sanitize algorithm in order to decide which signatures should be infeasible to compute without the private signing key.

Note that the unforgeability requirement implies that only the censor is able to change the message while maintaining the signing value constant. Therefore, a signer can prove to a judge the involvement of the censor in producing a sanitized message, by showing both the sanitized and the original messages and their common signing value. This implies that *accountability* follows from the unforgeability requirement.

We now proceed to define unforgeability via an adversarial game framework.

Unforgeability as an Adversarial Game Let \mathcal{A} be an algorithm that seeks to forge signatures. We assume that \mathcal{A} has oracle access to the SIGN as well as to the SANITIZE algorithms.

The sign oracle $\mathcal{O}^{sk_{sign}}$ is initialized with a positive integer q_a which indicates the number of queries it will accept during the period of the experiment. Similarly, the sanitize oracle $\mathcal{O}^{sk_{sanit}}$ is initialized with integer q_b , the maximum number of queries it will answer. Either oracle, if its quota of queries has been exhausted, answers all further queries with the special symbol \perp .

A sanitizable signature scheme is *unforgeable* if every efficient adversary has negligible probability of success in the following 2-phase experiment. Given a security parameter, and a pair of signing and sanitizing public keys, the adversary can interact with the associated SIGN and SANITIZE oracles. At the end of the first phase the adversary outputs a state (representing the knowledge acquired during the first phase) and a message m of its choice. In the second phase, the adversary again interacts with the oracles SIGN and SANITIZE, and its output is a candidate signature σ . The adversary wins if m was not queried to either the SIGN or SANITIZE oracles during either phase of the experiment, and if σ is a valid sanitizable signature on m . The advantage of the adversary is computed as its success probability over all instances of size k and random choices made by the adversary.

We say that a sanitizable signature is $(\epsilon, k, q_a, q_b, t)$ -*unforgeable* if for all probabilistic algorithms running in at most t steps, making no more than q_a queries to the SIGN algorithm and no more than q_b queries to the SANITIZE algorithm has probability of success smaller than ϵ on problem instances of size k .

4 Construction Based on Chameleon Hashes

In this section we provide a construction of sanitizable signatures based on chameleon hashes presented in [1]. We follow the well-established encode-and-sign paradigm and construct a generalized signature scheme compatible with standard signature schemes (e.g. RSA or DSS). As with any digital signature scheme, a sanitizable signature scheme needs to bind the signer to the message signed, thus providing non-repudiation. Our sanitizable signature schemes are practical and efficient.

Chameleon signatures were introduced by Krawczyk and Rabin [25], and in turn are related to the notion of undeniable signatures [9, 13, 14].

4.1 Setup

The parties involved are: A signer S with public and private keys (pk_{sign}, sk_{sign}) associated with the signature scheme, a (semi-trusted) censor C with public and private keys (pk_{sanit}, sk_{sanit}) associated with a chameleon hashing scheme, a verifier V , and a judge J (trusted third party).

Our construction consists of the following components:

- A secure digital signature scheme with signature, $s(\cdot)$, and verification, $v(\cdot)$, operations. We employ any standard signature scheme with any hash-and-encode mechanisms, such as RSA-EMSA-PSS [7, 35]. Note that we use SIGN

- for the sanitizable signature and $s(\cdot)$ for the underlying signature algorithm to avoid confusion, and similarly for VERIFY and $v(\cdot)$. The notation $S_{sk}(m, r)$ stand for the output of the basic signature algorithm applied to the value of an hash-and-encode function with input m ; if the signature scheme is probabilistic, the optional value r indicates the auxiliary random coins r .
- A chameleon hashing scheme [25, 1]. A chameleon hash computed over a message m with randomness r , and under public key pk will be denoted by $CH_{pk}(m, r)$. A chameleon hash (or *trapdoor commitment*) has the same properties of any cryptographic hash function and, in particular, it provides collision resistance. However, the owner of the private key sk corresponding to the public key pk can find collisions, i.e., messages m' such that $CH_{pk}(m, r) = CH_{pk}(m', r')$. By definition, chameleon hashes are always probabilistic algorithms, and to verify the correctness of a computed chameleon hash value C it is necessary to provide both the original message m and the randomness r used.

4.2 Sanitizable Signing

Suppose we wish to sign a document $m = (m_1, \dots, m_t)$ that is partitioned into t blocks, for some constant t . First, the signer selects a random unique document identifier ID_m and decides which portions, say m_{i_1}, \dots, m_{i_k} , of the document can be modified by the censor with public key pk_{sanit} . This allows the signer to compute a chameleon hash, denoted by $CH_{pk_{sanit}}(\cdot)$, under the censor's public key, on those portions of the message:

$$\sigma = SIGN(m, r; sk_{sign}, pk_{sanit}) := S_{sk_{sign}}(ID_m || t || pk_{sanit} || \bar{m}_1 || \dots || \bar{m}_t),$$

where $\bar{m}_i = CH_{pk_{sanit}}(ID_m || i || m_i, r_i)$ for $i \in \{i_1, i_2, \dots, i_k\}$, otherwise $\bar{m}_i = m_i || i$. The value r should be interpreted as the concatenation of all the random coins $r_{i_k}, i = 1, \dots, k$. In order to verify the above signature, one needs σ , m , r , and *auxiliary information to allow for segmentation of m into blocks*.

The length of the sanitizable signature is proportional to the number of *mutable* message blocks only (that is, the number of chameleon hashes in the input), because the verification of each chameleon hash requires an auxiliary randomness parameter. We stress that the underlying signature scheme is computed on a single, fixed-length (e.g., 160-bit) value, the hash-encoding of the concatenated input.

Because only the censor knows the private key corresponding to pk_{sanit} , it only can find collisions of the chameleon hash with arbitrary message blocks substituting for the original message block values. In particular, the censor can produce triples (ID_m, i, m'_i) such that:

$$CH_{pk_{sanit}}(ID_m || i || m_i, r_i) = CH_{pk_{sanit}}(ID_m || i || m'_i, r'_i).$$

Notice that the signer can prove that it did not generate a signature on a sanitized message by revealing the original message to a trusted third party (the judge). The fact that a collision of the chameleon hash exists implies that the

ensor has sanitized the document (only the censor can compute collisions). Note the use of the document identifier ID_m , and a block index. These are needed to prevent re-use of mutable blocks within a message or across messages that would enable changing of documents without censor intervention by re-use of sanitized blocks.

4.3 Chameleon Hash

It is important to remark that not all the chameleon hashes are suitable for our construction. For instance, the chameleon hash defined in [25] on a pair (m, r) is of the form $CH_y(m, r) = y^m g^r$, where $y = g^x$ and g is the generator of a prime order cyclic group and x is the private key. If the original message is sanitized and transformed into (m', r') then the signer can recover the private key x . Indeed, from $g^m y^r = g^{m'} y^{r'}$, x can be computed as $x = \frac{m' - m}{r - r'}$.

This *key exposure* problem was first addressed in [2], where a partial solution via identity-based constructions is proposed, and fully explored in [16, 1]. In particular, in [1] a *strongly unforgeable*[‡] chameleon signature scheme is provided, with the property that no trapdoors are ever revealed through collisions. This is in contrast with other constructions in [2, 16, 1], where at least an ephemeral trapdoor is compromised with each forgery.

Since our sanitizable signature construction requires strongly unforgeable chameleon hashes, it must use the scheme introduced in [1], which is related to a twin Nyberg-Rueppel signature [29, 28]. The scheme specifies a prime [§] p of bitlength κ , i.e., $p = uq + 1$, where q is also prime, and a generator g of the subgroup of squares of order q . The private key x is selected at random in $[1, q - 1]$, and the public key is $(g, y = g^x)$. Let \mathcal{H} be a (traditional) collision-resistant hash function, mapping arbitrary-length bitstrings to strings of fixed length τ : $\mathcal{H} : \{0, 1\}^* \rightarrow \{0, 1\}^\tau$.

To commit to a message m , it is sufficient to choose randomness $r = (\rho, \delta) \in \mathbf{Z}_q \times \mathbf{Z}_q$, and compute (cf. [1]):

$$e = \mathcal{H}(m, \rho); \text{ and } CH_y(m, \rho, \delta) = \rho - (y^e g^\delta \pmod p) \pmod q.$$

While the commitment can be computed by any party, the computation of a collision requires knowledge of the private key x , as follows. Let C denote the output of the chameleon hash on input $(m, r) = (m, \rho, \delta)$. First, a random value $k' \in [1, q - 1]$ is generated and then the other values are computed as: $\rho' = C + (g^{k'} \pmod p) \pmod q$, $e' = \mathcal{H}(m', \rho')$, and $\delta' = k' - e'x \pmod q$. Notice that indeed:

$$\rho' - (y^{e'} g^{\delta'} \pmod p) \pmod q = C + (g^{k'} \pmod p) - (g^{xe'} g^{\delta'} \pmod p) \pmod q = C.$$

Therefore, $(m', r') = (m', \rho', \delta')$ is the sought collision.

[‡] This terminology is not used in [1], but we adopt it here as it is related to the strong unforgeability of signature schemes.

[§] For conciseness of description, we discuss the Nyberg-Rueppel signature in the classical setting \mathbf{Z}_p^* . However, the same scheme can be defined over elliptic curves and would have better performance at comparable security settings.

4.4 Security Requirements

Correctness: It is clearly achieved, since the SIGN and VERIFY algorithms are modifications of a basic signature scheme, wherein mutable message blocks have been substituted by chameleon hashes.

Indistinguishability: In [1], it is shown that the chameleon hash based on the twin Nyberg-Rueppel signature provides *semantic security*, i.e., it is impossible to distinguish the distributions

$$\mathcal{S}_1^y = \{(m_1, r, C); C = CH_y(m_1, r)\} \text{ and } \mathcal{S}_2^y = \{(m_2, r, C); C = CH_y(m_2, r)\}.$$

This is exactly the same requirement for indistinguishability of sanitizable signatures. It is straightforward to verify that this semantic security furthermore implies privacy.

Identical distribution of sanitized and original signatures: The sanitization algorithm invokes the trapdoor collision-finding algorithm of the chameleon hash, in effect obtaining an alternative set of inputs to the sign algorithm that evaluate to the same signing value. Moreover, the outputs of the chameleon hash are statistically independent of the input message – again, see [1], and the proof for the semantic security property. The output distributions for SIGN and SANIT are therefore identical, and from that it follows that changes to the mutable parts of the message are undetectable (weak transparency).

Unforgeability: Our proof works by contradiction. Assuming the existence of an efficient adversary that defeats our chameleon-hash based sanitizable signature construction we show how to construct either an efficient algorithm to break the underlying signature scheme, or an efficient algorithm to compute chameleon hash collisions. The proof is straightforward but lengthy so we have postponed it to appendix §A.

5 Extensions and other Constructions

One natural extension is to allow for multiple censors, each able to modify different portions of the document. To achieve this, one may simply list all the public keys in the argument to the signature (and use each public key for the chameleon hash of the corresponding message block):

$$\begin{aligned} \text{SIGN}(m, r; sk_{\text{sign}}, pk_{\text{sanit}}^1, \dots, pk_{\text{sanit}}^t) := \\ S_{sk_{\text{sign}}}(ID_m || t || pk_{\text{sanit}}^1 || \dots || pk_{\text{sanit}}^t || \bar{m}_1 || \dots || \bar{m}_t). \end{aligned}$$

A different extension is to allow for distributed, threshold-trust censors. This can be easily achieved by using a threshold version of the chameleon hashing scheme.

A more interesting extension is to support *strong transparency*. One way to accomplish this would be for the signer to use the multiple-censor extension

described above, declare every block of the message mutable, but assign public keys of non-existing (dummy) sensors to the blocks the signer wish to remain unmodified. Unfortunately, in practice it may be difficult to hide the information about which sensors are fictitious, since probably there will be only a few well-known sensors and any other public key would give rise to suspicion of non-existence.

5.1 Hybrid Scheme

The construction described below is an extension of the redactable signature schemes, discussed in [23], based on the Gennaro-Halevi-Rabin signature [17]. It can be seen as an *improved* redactable signature of constant size which combines the advantages of both redactable and sanitizable signatures. In particular, the signature allows message blocks to be redacted by anyone while unredacted blocks can be sanitized by a censor.

The signature in [17] requires an RSA-type modulus n which is the product of two *safe* primes, p and q , that is, such that $(p - 1)/2$ and $(q - 1)/2$ are also primes. The public key is (v, n) for a randomly selected $v \in \mathbf{Z}_n^*$. To sign a message m , first compute the hash of it $\mathcal{H}(m)$ and then release y such that $y^{\mathcal{H}(m)} = v \pmod n$.[¶]

In [23], the following method is described to compute redactable signatures on a document $x = (x_1, \dots, x_k)$: First generate a document identifier ID_x and then release the signature $(ID_x || y)$ where $y = v^{1/(\mathcal{H}(ID_x || 1 || x_1) \times \dots \times \mathcal{H}(ID_x || k || x_k))} \pmod n$. As reported in [23], to redact the message block x_i it is sufficient to release the new signature $(ID_x || y')$ where $y' = y^{\mathcal{H}(ID_x || i || x_i)}$.

To make the redactable signature above sanitizable, we simply replace each triple $\mathcal{H}(ID_x, i, x_i)$ that can be sanitized with $\mathcal{H}(ID_x || i || \text{CH}_y(ID_x || i || x_i))$, that is each message block x_i is replaced with a chameleon hash of it computed under the public key of the censor. Now the censor will be able to modify the i^{th} block and produce a valid sanitized signature. Note that the proof of security in [23] still holds because the outer hash $\mathcal{H}(\cdot)$ remains unchanged.

5.2 Attribute Tags

Certain applications may require the censor to modify mutable parts of the message so that the new parts satisfy prescribed semantics or policies. For instance, the censor could replace an address only with a generic geographic location, an exact date only with a time period, an integer only with another integer in a specific range, or a certain age with “senior” or “minor,” and so on.

A simple solution is to prepend an immutable attribute tag to a mutable section of the message and expect the verifier to check that the data type of the mutable portion matches the specifications of the prepended attribute tag. The original signer could, for instance, prepend to a mutable part the phrase “Address (or area):” and make it immutable. In this way the verifier of the signature will

[¶] Note that $\gcd(\mathcal{H}(m), \phi(n)) = 1$ with overwhelming probability.

expect after that phrase either an address or a geographic location. Clearly, with this method, the original signer can specify the type of the mutable part and which conditions it should satisfy. For instance, the immutable phrase “Value (integer in $[0, 100]$):” indicates that the next mutable value must be an integer and in the range from 0 to 100.

6 Implementation

We implemented our basic sanitizable signature construction with the Nyberg-Rueppel-based chameleon hash, and performed a series of experiments to demonstrate the efficiency of sanitizable signatures.

6.1 Experiment Setup

Our implementation incorporates OpenSSL 0.9.7e Library routines. The code is compiled with gcc 3.4.2 (Red Hat). All tests are run under Fedora Core 3 with Linux 2.6.9 kernel on Pentium-4 2.6-GHz PC with 512 MB of RAM.

The 1024-bit keys used for RSA signatures are generated using OpenSSL’s command-line RSA key generation routine. We used OpenSSL Diffie-Hellman library routines to generate our 1024-bit Nyberg-Rueppel key using 5 as the generator (OpenSSL is optimized for 2 or 5 as the generator). Unfortunately, one cannot store the key as Diffie-Hellman parameters because OpenSSL does not write DH keys to file. So we store the Nyberg-Rueppel keys in DSA format.

In our implementation we chose hash-and-sign RSA as the generic signature algorithm and SHA-1 as the generic hash algorithm. Notice that hash-and-sign RSA is not secure but we are using it just as a lower-bound for our performance measurements. In a real scenario, a secure hash-and-encode scheme should be used, such as EMSA-PSS [7, 35].

We applied our implementation on two 1 KB random message blocks. The first block is the modifiable portion of the document; the second block is the fixed portion. To generate a sanitizable signature, we apply the Nyberg-Rueppel-based chameleon hash to the first block, concatenate the result to the second message block, and finally apply hash-and-sign RSA signature. We used 128-bit labels to serve as message block IDs. Signing and verifying both use OpenSSL RSA signature routines.

6.2 Results

We applied each specific operation 1000 times. The average performance results from our experiments, where the amount of time specified is for a single operation, are summarized in Table 1. These results show that the execution time for each operation we tested is of the order of 10 milliseconds. Hence the Nyberg-Rueppel-based sanitizable signature scheme is practical and efficient. Furthermore, sanitizable signing costs about four times the signing time of RSA

signature with SHA-1, while providing significant subsequent advantages in a setting where sanitization is required.

While sanitizable Nyberg-Rueppel verification is faster than signing, its relative performance vis-a-vis RSA-SHA-1 verification is worse. This results from verification being approximately 10 times faster than signing for RSA signatures, while only about 1.3 times faster for sanitizable Nyberg-Rueppel.

We note that the OpenSSL library, while implementing several optimizations for the RSA cryptosystem, does not include optimized code for discrete logarithm constructions. In particular, it does not support optimizations for simultaneous multiple exponentiation, as described in [20], and its performance is an order of magnitude slower than libraries such as Crypto++ [11].

SHA-1	RSA_{sign}	RSA_{verify}	CH_{NR}	CH_{NR} (Collision)	$SIGN$	$VERIFY$
0.027 ms	10.653 ms	0.609 ms	33.863 ms	28.196 ms	44.518 ms	34.497 ms

Table 1. Performance of OpenSSL primitives and Nyberg-Rueppel sanitizable signature algorithms.

7 Conclusions

Sanitizable signatures allow a semi-trusted censor to modify designated portions of a document and then produce a valid signature of the legitimately modified document without help from the signer. Moreover, a verifier cannot determine whether a received signature has been sanitized by the censor. We have implemented the scheme and the performance results obtained demonstrate that the scheme is practical and efficient.

Acknowledgments: We are grateful to Aniello Del Sorbo for helping with configuration issues of OpenSSL. We thank the anonymous referees of ESORICS 2005 for their insightful comments. This work was partially supported by NSF.

References

1. G. Ateniese and B. de Medeiros. On the key-exposure problem in chameleon hashes. *Proceedings of the Fourth Conference on Security in Communication Networks (SCN'04)*, Lect. Notes Comp. Sci., vol. 3352. Springer-Verlag, 2005. Full version: Cryptology ePrint Archive, Report 2004/243, <http://eprint.iacr.org/2004/243>
2. G. Ateniese and B. de Medeiros. Identity-Based Chameleon Hash and Applications. In Ari Juels, ed., *Proc. of Financial Cryptography (FC 2004)*, Lect. Notes Comp. Sci., vol. 3110, pp. 164–180. Springer-Verlag, 2004
3. M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography: the case of hashing and signing. In Y. Desmedt, ed., *Advances in Cryptology-CRYPTO '94*, Lect. Notes Comp. Sci., vol. 839, pp. 216-233. Springer-Verlag, 1994.

4. M. Bellare, O. Goldreich, and S. Goldwasser. Incremental cryptography with application to virus protection. In *Proc. of the Twenty-Seventh Annual ACM Symposium on Theory of Computing (FOCS'95)*, pp. 45–56. ACM Press, 1995.
5. M. Bellare and D. Micciancio. A new paradigm for collision-free hashing: Incrementality at reduced cost. In *Advances in Cryptology–Eurocrypt'97*, Lect. Notes Comp. Sci., vol. 1233. Springer-Verlag, 1997.
6. M. Bellare and G. Neven. Transitive signatures based on factoring and RSA. In Y. Zheng, ed., *Advances in Cryptology–ASIACRYPT'02*, Lect. Notes Comp. Sci., vol. 2501, pp. 397–414. Springer-Verlag, 2003.
7. M. Bellare, P. Rogaway. PSS: Provably secure encoding method for digital signature. IEEE P1363a: Provably secure signatures. <http://grouper.ieee.org/groups/1363/p1363a/pssigs.html> (1998)
8. M. Bishop, B. Bhumeratana, R. Crawford, and K. Levitt. How to Sanitize Data. Proceedings of the 13th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE-2004). Pp. 217-222. June 2004, Modena, Italy.
9. J. Boyar, D. Chaum, I. B. Damgård, T. P. Pedersen. Convertible undeniable signatures. In *Advances in Cryptology–CRYPTO'90*, Lect. Notes Comp. Sci., vol. 537, pp. 189–205. Springer-Verlag, 1990.
10. M. Burmester and T. van Le. Secure communications in Ad-hoc networks. In *Proc. of the 5th IEEE Information Assurance Workshop (IAW'05)*, pp. 234–241. 2004.
11. Crypto++ Library 5.2.1. <http://www.eskimo.com/weidai/cryptlib.html>
12. S. Chari, T. Rabin, and R. Rivest. An efficient signature scheme for route aggregation. Unpublished manuscript, 2002. <http://theory.lcs.mit.edu/rivest/publications.html>
13. D. Chaum. Zero-knowledge undeniable signature. In *Advances in Cryptology–EUROCRYPT'90*, Lect. Notes Comp. Sci., vol. 473, pp. 458–464. Springer-Verlag, 1990.
14. D. Chaum and H. Antwerpen. Undeniable signatures. In *Advances in Cryptology – CRYPTO'89*. Lect. Notes Comp. Sci., vol. 435, pp. 212–216. Springer-Verlag, 1991.
15. D. Chaum, E. van Heijst, B. Pfitzmann. Cryptographically strong undeniable signatures, unconditionally secure for the signer. In *Advances in Cryptology–CRYPTO'91*, Lect. Notes Comp. Sci., vol. 576, pp. 470-ff. Springer-Verlag, 1991.
16. X. Chen, F. Zhang, and K. Kim. Chameleon hashing without key exposure. In *Proc. of the 7th International Information Security Conference (ISC'04)*, Lect. Notes Comp. Sci., vol. 3225, pp. 87–98. Springer-Verlag, 2004.
17. R. Gennaro, S. Halevi, and T. Rabin. Secure hash-and-sign signatures without the random oracle. In *Advances in Cryptology–EUROCRYPT'99*, Lect. Notes Comp. Sci., vol. 1592, pp. 123–139. Springer-Verlag, 1999.
18. S. Haber and W. S. Stornetta. How to Time-Stamp a Digital Document. In *Advances in Cryptology–CRYPTO'90*, Lect. Notes Comp. Sci., vol. 537, pp. 437–455. Springer-Verlag, 1990.
19. Hakan Hacigumus, Balakrishna R. Iyer, and Sharad Mehrotra. Executing SQL over encrypted data in the database-service-provider model. In *Proc. Intern. Conf. Management of Data (ACM SIGMOD 2002)*, pp. 216–227. ACM Press, 2002.
20. A. J. Menezes, P. C. van Oorschot and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, 2001.
21. Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and Integrity in Outsourced Databases. In *Proc. of the Network and Distributed System Security Symposium (NDSS'04)*, 10 pp. Internet Society (ISOC) Press,

2004. <http://www.isoc.org/isoc/conferences/ndss/04/proceedings/Papers/-Mykletun.pdf>
22. D. Johnson and D. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks, Mobile Computing, 1996.
 23. R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic signature schemes. In B. Preneel, ed., *Topics in Cryptology-CT-RSA 2002*, Lect. Notes Comp. Sci., vol. 2771, pp. 244–262. Springer-Verlag, 2002.
 24. S. Kent, C. Lynn and K. Seo. Secure Border Gateway Protocol (Secure-BGP), IEEE Journal on Selected Areas in Communications, April 2000.
 25. H. Krawczyk and T. Rabin. Chameleon signatures. In *Proceedings of the Network and Distributed Systems Security Symposium (NDSS 2000)*, pp. 143–154.
 26. S. Micali and R. Rivest. Transitive signature schemes. In B. Preneel, ed., *Topics in Cryptology-RSA-CT'02*, Lect. Notes Comp. Sci., vol. 2271, pp. 236–243. Springer-Verlag, 2002.
 27. S. L. Murphy, M. R. Badger, and B. Wellington. OSPF with digital signatures. Internet Engineering Task Force (IETF) Request for Comments (RFC) 2154, June 1997.
 28. D. Naccache, D. Pointcheval, and J. Stern. Twin signatures: An alternative to the hash-and-sign paradigm. In P. Samarati, ed., *Proceedings of the Eighth Annual ACM Conference on Computer and Communications Security*, pp. 20–27. ACM Press, 2001.
 29. K. Nyberg and R. A. Rueppel. Message recovery for signature schemes based on the discrete logarithm problem. In *Designs, Codes, and Cryptography*, vol. 7(1–2), pp. 61–81. Kluwer Academic Publishers, 1996.
 30. R. Pang and V. Paxson. A High-level Programming Environment for Packet Trace Anonymization and Transformation. In Proc. ACM SIGCOMM 2003.
 31. R. Perlman. Network layer protocols with Byzantine robustness. Ph.D. thesis, Dept. of Elect. Eng. and Comp. Sci., Massachusetts Institute of Technology, August 1988.
 32. M. Peuhkuri. A method to compress and anonymize packet traces. In Proceedings of the ACM SIGCOMM Internet Measurement Workshop, November 2001.
 33. Y. Rekhter and T. Li. Border Gateway Protocol 4 (BGP-4), Internet Engineering Task Force (IETF) Request for Comments (RFC) 1771. March 1995.
 34. R. Rivest. Two signature schemes. Slides from talk given at Cambridge University, Oct. 17, 2000. <http://theory.lcs.mit.edu/~rivest/publications.html>
 35. RSA Labs: RSA Cryptography Standard: EMSAPSS – PKCS#1 v2.1. (2002)
 36. R. Steinfeld, L. Bull, and Y. Zheng. Content extraction signatures. In K. Kim, ed., *Information Security and Cryptology-ICISC'01*, Lect. Notes Comp. Sci., vol. 2288, pp. 285–304. Springer-Verlag, 2002.
 37. S. F. Shahandashti, M. Salmasizadeh, and J. Mohajeri. A provably secure short transitive signature scheme from bilinear group pairs. In C. Blundo and S. Cimato, eds., *Security in Communication Networks-SCN'04*, Lect. Notes Comp. Sci., vol. 3352, pp. 60–76. Springer-Verlag, 2005.
 38. United States of America Department of Health and Human Services. Standards for Privacy of Individually Identifiable Health Information: Final Rule, Federal Register: August 14, 2002, vol. 67, no. 157.
 39. J. Xu, J. Fan, M. Ammar, and S. B. Moon. On the design and performance of prefix preserving IP traffic trace anonymization. In Proceedings of the ACM SIGCOMM Internet Measurement Workshop, November 2001.

A Proof of unforgeability

Let \mathcal{A} be an $(\epsilon, k, q_a, q_b, t)$ -forgery algorithm defeating the security of our sanitizable signature construction (notation as in section §3.1); we show how to use this adversary to either undermine the security of the underlying signature scheme, or to find collisions for the chameleon hash signature scheme, in violation of their proven security properties.

Theorem 1. *Let \mathcal{A} be an $(\epsilon, k, q_a, q_b, t)$ -forger of a sanitized signature scheme. Then there exist an (ϵ', k, q_a, t') -forger of the underlying signature scheme and an $(\epsilon'', k, q_b, t'')$ -forger of the chameleon hash function, where the quantities are related by*

$$\epsilon \leq \epsilon' + \epsilon''; t \geq t' - q_b t_{\text{collision}}; t \geq t'' - q_a t_{\text{sign}},$$

where $t_{\text{collision}}$ and t_{sign} are, respectively, the maximum running times of the hash-collision finding and the signing algorithms on instances of size k .

Denote by μ the intermediate value such that $\sigma = S_{sk_{\text{sign}}}(\mu)$, i.e., μ is the value that is signed by the underlying signature algorithm in the process of sanitizable-signing m . Consider an instance of the forging experiment in which \mathcal{A} succeeds in computing a signature σ on a new message m , where $m = S_{sk_{\text{sign}}}(\mu)$. This instance must fall in (at least) one of two cases:

Case 1: Every query m' to the oracle $\mathcal{O}^{sk_{\text{sign}}}$ during \mathcal{A} 's execution resulted in signatures $\sigma' = S_{sk_{\text{sign}}}(\mu')$ associated to intermediate values μ' which are distinct from the value μ for the successful forgery $\sigma = S_{sk_{\text{sign}}}(\mu)$.

Case 2: There is a query m_i to the oracle $\mathcal{O}^{sk_{\text{sign}}}$ such that the response σ_i equals $S_{sk_{\text{sign}}}(\mu)$, with m_i different from m .

In the first case, proceed as follows to build an adversary \mathcal{B} of the underlying signature algorithm. First, \mathcal{B} generates a pair of public and private keys for the chameleon hash function, $(sk_{\text{sanit}}, pk_{\text{sanit}})$. It uses sk_{sanit} with the collision-finding algorithm for the chameleon hash function to emulate the oracle $\mathcal{O}^{sk_{\text{sanit}}}$, and it gives pk_{sanit} to the adversary \mathcal{A} . In order to answer \mathcal{A} 's signature queries, \mathcal{B} resorts to its own signing oracle for the underlying signature scheme. When \mathcal{A} finishes computing σ , \mathcal{B} outputs μ for its choice of target message; and the whole transcript of \mathcal{A} 's execution as its state after the first phase. (Note that μ is available from \mathcal{A} 's transcript otherwise the verification of \mathcal{A} 's success cannot be ascertained via the sanitized verification algorithm.)

In its second phase, \mathcal{B} just reads σ from the state information from the first phase, and terminates successfully whenever \mathcal{A} succeeds, and the execution is an instance of case (1). \mathcal{B} 's execution time equals $t' = t + q_b t_{\text{collision}}$, where t is the number of steps used by \mathcal{A} , q_b is the number of queries to the sanitization oracle, and $t_{\text{collision}}$ is the (maximum) number of steps executed by the hash-collision algorithm on instances of size k , which \mathcal{B} must perform to emulate answers to the sanitization oracle.

In the second case, algorithm \mathcal{A} could be used to build an adversary \mathcal{C} of the chameleon hash algorithm. First, \mathcal{C} generates a pair of public and private keys for the underlying signature algorithm (sk_{sign}, pk_{sign}) . It uses sk_{sign} with the underlying signing algorithm $s(\cdot)$ to emulate the signing oracle $\mathcal{O}^{sk_{sign}}$, and conveys pk_{sign} to the adversary \mathcal{A} . To answer \mathcal{A} 's sanitization queries, \mathcal{C} resorts to the collision-finding oracle for the strongly unforgeable chameleon hash function.

When \mathcal{A} finishes computing σ , \mathcal{C} retrieves the value μ and compares it with the values μ_i that appear in \mathcal{A} 's transcript of queries to the signing oracle. Since we are in case (2), there is at least one queried message m_i that differs from m but such that μ_i equals μ . Note that m can differ from m_i only if they differ in some mutable block (otherwise $\mu \neq \mu_i$). For simplicity of notation we assume that m and m_i are a single block each. Therefore, we have $C = CH_{pk_{sanit}}(m, \rho, \delta) = CH_{pk_{sanit}}(m_i, \rho_i, \delta_i)$, and \mathcal{C} outputs C, m_i, ρ_i, δ_i as its chosen value to seek collisions against, and the whole transcript of \mathcal{A} 's execution as its state after the first phase of the adversarial game.

In its second phase, \mathcal{C} just reads the values m, ρ, δ from the transcript of \mathcal{A} and outputs it. Therefore, \mathcal{C} succeeds whenever \mathcal{A} succeeds and \mathcal{A} 's execution is of type (2). \mathcal{C} 's execution time is $t'' = t + q_a t_{sign}$, where t is the number of steps used by \mathcal{A} , q_a is the number of \mathcal{A} 's queries to the signing oracle, and t_{sign} is the (maximum) number of steps executed by the underlying signing algorithm on instances of size k , which is executed to emulate the signing oracle. \square