Strongly Unforgeable Proxy Re-Signature Schemes in the Standard model

No Author Given

No Institute Given

Abstract. Proxy re-signatures are generally used for the delegation of signing rights of a user (delegator) to a semi-trusted proxy and a delegatee. The proxy can convert the signature of one user on a message into the signature of another user on the same message by using the delegation information (rekey) provided by the delegator. This is a handy primitive for network security and automated delegations in hierarchical organizations. Though proxy re-signature schemes that are secure in the standard model are available, none of them have addressed the security notion of strong existential unforgeability, where the adversary will not be able to forge even on messages for which signatures are already available. This is an important property for applications which involve the delegation of authentication on sensitive data. In this paper, we define the security model for strong unforgeability of proxy re-signature schemes. We propose two concrete strong unforgeable proxy re-signature schemes, where we induce the strong unforgeability in the scheme by embedding the transformation techniques carefully in the sign and resign algorithms. The security of both the schemes is related to the hardness of solving Computational Diffie-Hellman (CDH) problem.

1 Introduction

Proxy Re-Cryptography originally introduced by Blaze et. al. [4], has been an emerging field of interest in the research community. This has mainly been motivated by its applications and challenges faced in the construction of such schemes. Proxy Re-Signature and Proxy Re-Encryption schemes are the essential primitives in this field. Proxy Re-Encryption allows a semi-trusted proxy to convert a cipher text meant for a receiver into a cipher text of another user so that the other user can decrypt it using his secret key. Many such schemes have been designed and are being used for various applications like distributed storage, distributed rights management and cloud infrastructure.

Proxy Re-Signature scheme involves a semi-trusted proxy between two parties A and B, where the proxy has the capability and information (Re-Signature Key) to convert the signature on a message m by user A, into the signature by user B on the same message m [A-Delegatee, B-Delegator]. This kind of signature comes handy in situations where in B (the delegator) is not available to sign on a given message(m). Then using a signature of (the delegatee) A on the message(m), the semi-trusted proxy can generate a signature on behalf of B on the same message(m) with the help of a rekey(Re-Signature Key), without involving delegator B in the signing process. The ideal properties of a proxy re-signature scheme are unidirectional, multi-use, public proxy, transparency, key-optimal, non-interactive and non-transitivity which were standardized by [2]. The descriptions for these properties can be found in Appendix A.

Proxy Re-Signatures are generally proved for an unforgeability notion where an adversary will not be able to forge a signature/re-signature on a new message rather than on a message that has already been signed/resigned. But there might be applications with stricter security, which require the system to protect the existing message-signature pairs from being forged. The security notion for such a requirement is called strong unforgeability, which we discuss in detail later.

Motivation: Proxy re-signature schemes have various applications as listed by [2]. For example, they can be used in huge organizations for employee are delegated to sign on behalf of the organization. Applications involving checkpoints can make good use of multi-use proxy re-signatures. This primitive can aid the delegation of certifying temporary public keys, which will as a result in a reduced overhead in the PKI network.
The property of strong unforgeability can also be adapted by applications shown above for a more stringent security. Consider sensitive applications where the signatures play a crucial role, there can be passive adversaries polling the communication channel. Hence they can store every message and its corresponding signature or re-signature. Then at a later time, they can modify these signatures due to their weak unforgeability property and present a new signature or re-signature for the same message at will.

Strong unforgeability in proxy re-signatures is an interesting property in practice. For instance, consider **access control mechanism** which involves delegation of access rights. Assume that the system wants to give access to a user-A with access policy "p", then the systems generates a signature S_A on "p" and gives it to the user-A. In the case when user-A and the system are unavailable to implement the policy when required, then there can be a middle system proxy to converts the signature S_A to S_B and gives it to user-B to perform the required action. In the case of revocation of access policies, neither user-A or user-B should be able to forge on the revoked signatures S_A and S_B and claim the access rights for "p".

We can adopt the same ideology for **intellectual property protection** in services like the digital media rental, where the owner of the copyright(sign) would give the content to the retailers(resigned) from whom the customers can rent the media. The copyright materials (embedded with the signature of the owner or the re-signature of the retailer) must be unforgeable by an external adversary which otherwise might lead to piracy. Hence the strongly unforgeable proxy re-signature can come in handy for such a situation where the pirated copies cannot possess a valid publically verifiable signature apart from its original copy. It can be used in conjunction with proxy re-encrpytion in the digital rights management.

Note: Further we would like to emphasize that it is not trivial to attain strong unforgeability in proxy re-signatures as it is different from that for the notion in signatures. The attacker must not be able to modify the signatures/re-signatures to belong another entity which can be publicly verified.

Related Work: Proxy Re-Signatures originally introduced by Blaze et. al. [4], is an interesting class of signature schemes. It was later formalized by Ateniese et. al. [2] who also defined a suitable security model for proving its security and supplementing it with two concrete schemes (one bi-directional and the other uni-directional) both secure in the random oracle model. Shao et. al. [13] proposed the first proxy re-signature scheme, which was bi-directional and secure in the standard model, with a new perspective on the security (Static Corruption similar to that in proxy re-encryption schemes). Chow et. al [6] showed an insecurity in Shao's scheme and gave a new proxy re-signature scheme, which was secure in the standard model but at the cost of transparency. Libert et. al. [9] came up with a multi-use, uni-directional (Open problem left in [2]) and non-transparent proxy re-signature scheme that is secure in the standard model. Recently, Shao et. al. came up with a novel approach [15] for the first ID based multi-use proxy re-signatures. It is to be noted from Table 1 that none of the existing proxy re-signature schemes secure in the standard model satisfy the stronger notion of existential unforgeability [1], where an adversary will not be able to forge a previously signed or re-signed message. In this paper we address this issue by defining a security model for such a security notion and propose two concrete schemes for the same.

Table 1. Proxy Re-Signature Schemes in the Standard Model with properties comparison

Properties	Shao et. al.[13]	Libert et. al.[9]	Chow et. al.[6]	Our Scheme
Uni-Directional	No	Yes	Yes	Yes
Multi-Use	Yes	Yes	No	No
Private Proxy	Yes	Yes	Yes	Yes
Non-Interactive	No	No	No	No
Non-Transitive	No	Yes	Yes	No
Transparent	Yes	No	No	No
Temporary	No	No	Yes	No
Strongly Unforgeable	No	No	No	Yes

Our Contribution: In this paper, we first define a security notion for strongly unforgeable proxy re-signature schemes based on the static corruption security model defined by Shao et. al. [13]. Then, based on Waters' scheme [18] we propose two strongly unforgeable proxy re-signature schemes secure in the standard model. After reviewing the existing transformation techniques for converting existentially unforgeable signatures to strongly unforgeable ones, we choose the transformation technique proposed by [5] strong unforgeability transformation and generic transformation by [7] which uses chameleon hash function. The schemes are constructed using bilinear maps and their security is based on the Computational Diffie-Hellman (CDH) assumption. We also suggest some efficiency improvements for the schemes.
Paper Organization: The paper is organized as follows. In Section 2 we give the various definitions which are involved in constructing and proving the security of the scheme. Section 3 reviews the available strong unforgeability transformations techniques in the standard model. In Section 4 and 5 we propose two concrete strongly unforgeable proxy re-signatures and also provide suggestions for efficiency improvement. Conclusion is offered in Section 6. The Appendices A and B presents the formal security argument for both the schemes.

2 Definitions

2.1 Proxy Re-Signature

The proxy re-signature is a collection of probabilistic polynomial time algorithms (KeyGen, ReKey, Sign, Verify, Re-Sign):
(KeyGen, Sign) : These algorithms are taken from the underlying signature scheme, and hence retain all its functionality and properties. We are using the same key construct (pk-public key, sk-secret key) for the rest of the scheme.

ReKey : On input of the secret keys (sk_A, sk_B) , the re-key generation algorithm generates a re-key which is to be stored in the proxy. The re-signature key may be calculated through an interactive protocol, where the secret keys of A and B are used to compute the re-key. At the end of the protocol, the proxy will obtain the re-key $(rk_{A\to B})$ without gaining any information regarding the corresponding secret keys of A and B. This re-key $(rk_{A\to B})$ is used by the proxy to transform the signatures of user A to that of user B.
ReSign : Takes as input $(rk_{A\to B}, pk_A, m, \sigma_A)$ and it first verifies whether the given signature is that of user A by performing $Verify(pk_A, m, \sigma_A)$. If the Verify returns false, then the algorithm aborts and reports of an invalid signature. Otherwise, the transformation of the signature takes place using the re-key and the transformed signature $\sigma_B = ReSign(ReKey(sk_A, sk_B), pk_A, m, \sigma_A)$ is returned.
Verify : The signature σ_B on message m, when generated directly by user B, will be verified by the Verify algorithm of the underlying signature scheme. However, if σ_B is generated by the proxy, we may use a different algorithm.
Correctness : All the untransformed and transformed signatures will satisfy the Verify algorithm.
Verify1(m, σ_A, pk_A) = True
Verify2(m, ReSign(ReKey(sk_A, sk_B), pk_A, m, σ_A), pk_B) = True
Note: We are using two verification algorithms Verify1 and Verify2 in order to verify the signature and re-signature respectively. Depending on the transparency property of the scheme, if the output of the sign and re-sign algorithm are computationally indistinguishable, the Verify1 and Verify2 are one and the same. On the other hand, in the case of non-transparency where the signature and the re-signature are distinguishable, the Verify1 and Verify2 are different algorithms. For multi-use, non-transparent schemes, the Verify2 will be varying for each level of signature translation. Hence Verify algorithm varies depending on the nature of the signature.

Security Model for Proxy Re-Signature Scheme
We present a security model for the proxy re-signature scheme, which ensures the strong unforgeability property. The security of the scheme defined here is inspired by Shao et al.'s security model [13] for existential unforgeability under static corruption. The same has also been discussed recently as an improved security model in uni-directional proxy re-signatures [14].
The security model is defined as the game between the forger F (the adversary trying to attack the system) and the challenger $\mathbb{C}$. The security is based on the ideology of static corruption where, a forger who is trying to attack the signature scheme determines the corrupted parties of the system prior to the start of game (between F and $\mathbb{C}$). It does not allow adaptive corruption of users of the system in between the security game. Hence our security model will deal with corrupted and uncorrupted parties accordingly. The security game is defined in the following phases:

Training Phase: The challenger simulates the following oracles, which the forger is allowed to query during this phase.

1. $O_{UKeyGen}$ Key Generation for Uncorrupted user : This oracle generates the key pair using the KeyGen and returns the public keys of uncorrupted users in the system.
2. $O_{CKeyGen}$ Key Generation for Corrupted user : This oracle generates the key pair using KeyGen and returns the public key and secret key of the corresponding corrupted user.
3. O_{ReKey} Re-Signature Key Generation : This oracle when given the public key of users A and B, generates a re-key $rk_{A\to B}$ only when both user A and B are either corrupted or uncorrupted. When one of them is corrupted and the other one is not, $\perp$ is returned.
4. O_{Sign} Signature : Given the input public key for user A which was generated by the one of the KeyGen oracles and a message m, the signature $\sigma_A(m)$ for the corresponding public key on the message is returned. The signature on the message is returned regardless of user A being a corrupted or uncorrupted user.
5. O_{ReSign} Re-Signature : Given the input $(pk_B, pk_A, m, \sigma_A)$, this oracle will return the corresponding transformed signature σ_B regardless of A or B being a corrupted or uncorrupted user. Notice that the rekey is not generated by $\mathbb{C}$, when one of the users (A or B) is corrupted. Still, this oracle must be able to return the re-signature.

This ensures that the forger gets all his training by querying the oracles (polynomial number of queries with respect to the security parameter) available to any user in the system. It is inherent that the forger controls the corrupted users. After the training phase, the strong unforgeability is captured in the following phase which exposes the true potential of the forger.

Forgery Phase : The forger after the training phase will return the forgery (m^*, pk^*, σ^*). The forgery is said to be a valid one if the following conditions are satisfied,

1. Verify algorithm must satisfy for (m^*, pk^*, σ^*) where σ^* can either be a transformed or untransformed signature.
2. pk^* used for forgery must be uncorrupted and whose keys are generated by the uncorrupted key generation oracle.
3. σ^* is not the output during the training phase with (m^*, pk^*) as input to the sign oracle.
4. σ^* is not the output of the ReSign oracle with input as (pk^*, pk, m^*, σ) during the training phase, for any pk and σ.

Note: When we consider the strong unforgeability for proxy re-signatures, we must take into account the fact that any re-signature must always be generated using the re-signature key. The delegator must not be able to exploit his own signatures in order to convert them to a re-signature.

2.2 Bilinear Pairing

Let $\mathbb{G}$ be an additive cyclic group generated by P, with prime order p, and $\mathbb{G}_1$ be a multiplicative cyclic group of the same order p. A bilinear pairing is a map $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$ with the following properties.

- **Bilinearity.** For all $P, Q, R \in \mathbb{G}$,
  - $\hat{e}(P + Q, R) = \hat{e}(P, R)\hat{e}(Q, R)$
  - $\hat{e}(P, Q + R) = \hat{e}(P, Q)\hat{e}(P, R)$
  - $\hat{e}(aP, bQ) = \hat{e}(P, Q)^{ab}$
- **Non-Degeneracy.** There exist $P, Q \in \mathbb{G}$ such that $\hat{e}(P, Q) \neq I_{\mathbb{G}_1}$, where $I_{\mathbb{G}_1}$ is the identity in $\mathbb{G}_1$.
- **Computability.** There exists an efficient algorithm to compute $\hat{e}(P, Q)$ for all $P, Q \in \mathbb{G}$.

Note: We will be using the multiplicative notation for our schemes, as it is easier to represent the standard model schemes [18] in this form. But it is to be understood that, $\mathbb{G}$ is basically an additive prime order group for our schemes.

2.3 Computational Diffie-Hellman Assumption

Computational Diffie-Hellman Problem: Let $\mathbb{G}$ be a group of prime order p and g be the generator of $\mathbb{G}$. The CDH problem can be defined as follows: An algorithm A is said to have an advantage ϵ in solving the CDH problem if

$$Pr[A(g, g^a, g^b) = g^{ab}] \geq \epsilon$$

where the probability is calculated over the random choices of a,b$\in \mathbb{Z}_p^*$, g $\in \mathbb{G}^*$ and the random bits used by algorithm A.

3 Strong Unforgeability Transformation

In this section we give an overview of the strong unforgeability property in signature schemes and provide our intuition regarding how we select and use the strong unforgeability transformation techniques for our constructions.
Generally signature schemes are proven for unforgeability under adaptive chosen message attack (The forger can adaptively choose a message for forgery). There are two kinds of security notions under this namely, strong and weak unforgeability. The most commonly used notion is the weak (existential) unforgeability, where the forgery is on a message that has never been queried for a signature, during the training phase.
This case is implicit for deterministic signature schemes where a message cannot have two different signatures with respect to a user. But, with the advent probabilistic signature schemes, the situation arises where a message can have one or more signatures with respect to a single user. Hence a stronger notion of security is required, where the forgery on a message which has already been signed (queried during the training phase) is also considered a valid attack on the system.
Let the forger have the following message signature pairs from the training phase:
(m_1, σ_1), (m_2, σ_2), ... , (m_q, σ_q). where q is the number of queries in the training phase.

- Weak forgeability: Here a forgery is on a message m^* where $m^* \notin \{m_1, m_2, ..., m_q\}$.
- Strong forgeability: Here, the forger may come up with a forgery (m^*, σ^*) where $(m^*, \sigma^*) \notin \{(m_i, \sigma_i) | 1 \leq i \leq q\}$. Note that, in the training phase, some of the queries may have m^* as the message.

While the technique in [16] can be applied for our construct, we stick to the basic transform used in [5], so that the efficiency does not reduce considerably.
The generic transformation using chameleon hash functions proposed by [7] can be applied to [18] for a strongly unforgeable signature. A concrete strongly unforgeable proxy re-signature scheme based on this transformation is given in section 4. While there are other transformation techniques [17, 11, 3], to obtain strong unforgeability, we avoid their usage as they compromise on the efficiency and the flexibility required to obtain a proxy re-signature.
Remark: It is to be noted that the techniques discussed above, cannot be applied directly to existing standard model proxy re-signature schemes as they may not satisfy the strongly unforgeability notion in its true sense. Hence it is required to perform a customized transformation in order to obtain a strong unforeability property. We illustrate a few insecurities in appendix C.

4 Scheme-1

In this section, we will present our first strongly unforgeable proxy re-signature scheme, which can be proved secure in the standard model. We name the scheme PRS_{SUF} , which is bidirectional, and single-use in nature. The construction of the scheme uses bilinear maps. The scheme uses the Boneh et. al.'s transformation technique and carefully adding extra randomness and certain parameters so that the proxy re-signature can remain strongly unforgeable from all aspects. The use of two key pairs for every user is justified by proving that the re-signature cannot be produced unless it is processed by the re-signature algorithm.
The input message to be signed can be of any size. This message will then be modified to a n-bit format before computing the signature or re-signature.

KeyGen(1^k): On the input of security parameter k, this algorithm performs the following computations to generate the keys of a user. Let $\mathbb{G}, \mathbb{G}_1$ be groups of prime order p. Let $\hat{e}$ be an admissible bilinear map where $\hat{e} : \mathbb{G} \times \mathbb{G} \rightarrow \mathbb{G}_1$. We consider a collision resistant hash function which is defined as $H_1 : \{0,1\}^* \Rightarrow \{0,1\}^n$. Since the scheme is in the standard model, the hash functions can be instantiated with standard hash functions which can be implemented in the real world. Here the length(n) of the message digest (Output of the H_1) depends on the security parameter.
Let g a generator for group $\mathbb{G}$ and random elements $< g_2, h, u_0, u_1 ... u_n, v_0, v_1 .. v_n > \in G^{2n+4}$.
The system(trusted party) then fixes this as a common reference string for all users, with which they can generate their respective public and secret key components. Consider a user A, who chooses $a_1, a_2 \in_R \mathbb{Z}_p$ and sets $g_1 = g^{a_1}$ and $g_{11} = g^{a_2}$ as his public keys. The secret key components computed for the user is

$$sk =< sk_1, sk_2 >=< g_2^{a_1}, g_2^{a_2} > .$$

Sign(m, sk): On the input of the message m to be signed and secret key of the signer, the signature algorithm performs the following computations

- Let $r, s \in_R \mathbb{Z}_p$
- Compute $\sigma_2 = g^r \in \mathbb{G}$
- Set $\sigma_3 = s \in \mathbb{Z}_p$
  $t_1 = H_1(m, \sigma_2) \in \{0,1\}^n$
- Set $m^{(1)} = H_1(g^{t_1} h^s) \in \{0,1\}^n$.
  Define $m_u^{(1)} = u_0 \prod_{i=1}^{n} u_i^{m_i^{(1)}}$
  where $m^{(1)} = (m_1^{(1)}, m_2^{(1)}, m_3^{(1)} \ ... \ m_n^{(1)})$ denotes the n-bit representation of $\mathrm{m}^{(1)}$.
- Compute $\sigma_1 = sk_1 . \left(m_u^{(1)}\right)^r = g_2^{a_1} . \left(m_u^{(1)}\right)^r \in \mathbb{G}$
  Thus, the signature $\sigma =< \sigma_1, \sigma_2, \sigma_3 >$ on the message m by the signer A with the secret key $g_2^{a_1}$ is given by

$$(\sigma_1, \sigma_2, \sigma_3) = \left(g_2^{a_1} . \left(m_u^{(1)}\right)^r, g^r, s\right)$$

Remark: The transformation [5] can be observed in $m^{(1)}$ which is used as the message component in the sign algorithm.

ReKey$(g_2^{a_1}, g_2^{b_1}, g_2^{b_2})$: In order to delegate the signing right from either A to B (or B to A as our scheme is bidirectional) the proxy will run an interactive protocol with A and B. The final re-signature key obtained by the proxy will be of the form

$$rk_{A \rightarrow B} = g_2^{b_1 + b_2 - a_1} \in \mathbb{G}$$

The Interactive protocol for using the secret keys of the users in a secure manner to calculate the final re-signature key is defined as follows:

1. The proxy initially chooses a random $\mathrm{R} \in \mathbb{Z}_p$, compute g_2^R and send it to user A.
2. Then A uses its secret key parameter a, computes and sends $g_2^R . g_2^{-a_1} = g_2^{R - a_1}$ to user B. B is chosen by A, as he is the user to whom A wishes to convert his signatures.
3. B uses his secret parameters b_1 and b_2, computes and sends $(g_2^{R-a_1}) . g_2^{b_1+b_2} = g_2^{R+b_1+b_2-a_1}$ to the proxy.
4. The proxy now computes $(g_2^{R+b_1+b_2-a_1}) . g_2^{-R} = g_2^{b_1+b_2-a_1}$ as the re-signature key.

Remark: The rekey algorithm is performed during the system setup through a secure channel or in private. We claim that the above interactive protocol between the parties involved in delegations is unavoidable because it involves the use of their secret information. Since it performs a secure computation, a non-interactive zero knowledge will not serve the purpose in this scenario. This has been the approach used in all the proxy re-signatures with private proxy defined until now.

ReSign$(rk_{A \rightarrow B}, pk_A, m, \sigma)$: Proxy on receiving a signature $\sigma =< \sigma_1, \sigma_2, \sigma_3 >$ on the message m by user A associated with public key pk_A, re-signature key $rk_{A \rightarrow B}$ as input, performs the following to generate a signature on m by user B.

- First check whether the $Verify1(pk_A, m, \sigma)$ is satisfied, if not abort reporting an invalid signature. Otherwise perform the following steps to generate the re-signature.
- Assign $\hat{\sigma_2} = \sigma_2 = g^r \in \mathbb{G}$
- Let $r_1 \in_R \mathbb{Z}_p$ and compute g^{r_1} and assign $\hat{\sigma_3} = g^{r_1} \in \mathbb{G}$
- Assign $\hat{\sigma_4} = \sigma_3 = s \in \mathbb{Z}_p$
  $t_2 = H_1(m, g^{r_1}, g^r) \in \{0,1\}^n$
- Set $m^{(2)} = H_1(g^{t_2} h^{\sigma_3}) \in \{0,1\}^n$.
  The computation of $m^{(2)}$ is in accordance with the strong unforgeability transformation.

  Define $m_v^{(2)} = v_0 \prod_{i=1}^{n} v_i^{m_i^{(2)}}$
  where $m^{(2)} = (m_1^{(2)}, m_2^{(2)}, m_3^{(2)} \ ... \ m_n^{(2)})$ denotes the n-bit representation of $\mathrm{m}^{(2)}$.

– Compute $\hat{\sigma_1} = \sigma_1.rk_{A\to B}.\left(m_v^{(2)}\right)^{r_1} \in \mathbb{G}$

$$= g_2^{a_1}.\left(m_u^{(1)}\right)^{r}.g_2^{(b_1+b_2-a)}.\left(m_v^{(2)}\right)^{r_1}$$
$$= g_2^{b_1+b_2}.\left(m_u^{(1)}\right)^{r}.\left(m_v^{(2)}\right)^{r_1}$$

The Re-signature $\hat{\sigma} =< \hat{\sigma_1}, \hat{\sigma_2}, \hat{\sigma_3}, \hat{\sigma_4} >$ is given by,

$$(\hat{\sigma_1}, \hat{\sigma_2}, \hat{\sigma_3}, \hat{\sigma_4}) = \left(g_2^{b_1+b_2}.\left(m_u^{(1)}\right)^{r}.\left(m_v^{(2)}\right)^{r_1}, g^{r}, g^{r_1}, s\right)$$

Remark: The transformation [5] can be observed in $m^{(2)}$ which is used as the message component in the resign algorithm. This transformed message re-randomizes the existing randomness obtained from the signature algorithm.

Verify1(m, σ, pk): For verifying the signature $\sigma = (\sigma_1, \sigma_2, \sigma_3)$ on the message m by the user corresponding to public key pk, any verifier can perform the following validity check. Initially compute,

– $\hat{t_1} = H_1(m, \sigma_2) \in \{0,1\}^n$

– $m^{(1)} = H_1(g^{\hat{t_1}}h^{\sigma_3}) \in \{0,1\}^n$

– Verify whether the following equation satisfies

$$\hat{e}\left(\sigma_1, g\right) \stackrel{?}{=} \hat{e}(m_u^{(1)}, \sigma_2)\hat{e}(g_2, g_1)$$

if the above test holds, output *Valid* otherwise output *Invalid*.

Correctness for verification of signature (Verify1):

$$\hat{e}(\sigma_1, g) \stackrel{?}{=} \hat{e}(m_u^{(1)}, \sigma_2)\hat{e}(g_2, g_1)$$

Right Hand Side:

$$= \hat{e}(m_u^{(1)}, g^{r})\hat{e}(g_2, g^{a_1})$$
$$= \hat{e}((m_u^{(1)})^{r}, g)\hat{e}(g_2^{a_1}, g)$$

By Bilinearity property of the map e:

$$= \hat{e}(g_2^{a_1}(m_u^{(1)})^{r}, g)$$
$$= \hat{e}(\sigma_1, g).$$

Verify2$(m, \hat{\sigma}, pk)$: For verifying the re-signature $\hat{\sigma} = (\hat{\sigma_1}, \hat{\sigma_2}, \hat{\sigma_3}, \hat{\sigma_4})$ on the message m by the user corresponding to public key pk, any verifier can perform the following validity check on the transformed signature as follows,

– Compute $\hat{t_1} = H_1(m, \hat{\sigma_2}) \in \{0,1\}^n$
– $m^{(1)} = H_1(g^{\hat{t_1}}h^{\hat{\sigma_4}}) \in \{0,1\}^n$
– $\hat{t_2} = H_1(m, \sigma_2, \sigma_3) \in \{0,1\}^n$

– $m^{(2)} = H_1(g^{\hat{t_2}}h^{\hat{\sigma_4}}) \in \{0,1\}^n$

– $m_u^{(1)} = u_0\prod_{i=1}^{n} u_i^{m_i^{(1)}}$
– $m_v^{(2)} = v_0\prod_{i=1}^{n} v_i^{m_i^{(2)}}$
– Verify whether the following equation satisfies

$$\hat{e}(\hat{\sigma_1}, g) \stackrel{?}{=} \hat{e}(m_u^{(1)}, \hat{\sigma_2}).\hat{e}(m_v^{(2)}, \hat{\sigma_3}).\hat{e}(g_2, g_1).\hat{e}(g_2, g_{11})$$

if the above test holds, output *Valid* otherwise output *Invalid*.

Correctness for verification of re-signature (Verify2):

$$\hat{e}(\hat{\sigma_1}, g) \stackrel{?}{=} \hat{e}(m_u^{(1)}, \hat{\sigma_2}).\hat{e}(m_v^{(2)}, \hat{\sigma_3}).\hat{e}(g_2, g_1).\hat{e}(g_2, g_{11})$$

Right Hand Side:

$$= \hat{e}(m_u^{(1)}, g^r)\hat{e}(m_v^{(2)}, g^{r_1})\hat{e}(g_2, g^{b_1})\hat{e}(g_2, g^{b_2})$$
$$= \hat{e}((m_u^{(1)})^r, g)\hat{e}((m_v^{(2)})^{r_1}, g).\hat{e}(g_2^{b_1+b_2}, g)$$

By Bilinearity property of the map e:

$$= \hat{e}(g_2^{b_1+b_2}(m_u^{(1)})^r(m_v^{(2)})^{r_1}, g)$$
$$= \hat{e}(\hat{\sigma_1}, g).$$

Proof of Security: We prove the security of the scheme PRS_{SUF} using the following theorem. In this theorem, we prove that breaking the scheme is as hard as solving the CDH problem.

Theorem 1. *If there is an adversary, which can break the strongly unforgeable scheme PRS_{SUF} in polynomial time, by having q_s and q_{rs} queries to the sign and resign oracles and advantage (ϵ) with n as the size of the message, then the CDH problem can be broken with advantage $\epsilon^{''} \geq \epsilon/144q_s(q_s + q_{rs})(n+1)^2$*

1. The security proof of the PRS_{SUF} can be divided into two parts. The first one is where we prove the security of the underlying signature scheme by simulating a weaker form of the signature with the sign and resign oracles. Thus we prove the scheme to be secure for any message that was not queried during the training phase.
2. Once we have proved the existential forgery for the signature and re-signature scheme, we may apply transformation (modifying the message by binding it with a randomness, without affecting the internal structure of the signature or re-signature) and then derive the security for strong unforgeability. Since this transformation [5] uses the assumption of universal one way hash functions, the resulting strongly unforgeable scheme is also secure.

As mentioned in the model, the security is proved as game between the challenger $\mathbb{C}$ and forger F. F is trained with the working of the system through the following simulation by $\mathbb{C}$. We are simulating the training phase for the weaker form of the re-signature(without transformation) where $m^{(1)}$ or $m^{(2)}$=m and prove for its existential unforgeability prior to applying the strong unforgeability transformation.
$\mathbb{C}$ is given the input of the hard problem, g^a, g^b and is required to find the solution g^{ab} from the forgery performed by the forger. Hence $\mathbb{C}$ embeds the hard problem instances while simulating the system to the adversary, which is described in detail in Appendix A.

5 Scheme-2

Recently [7], proposed a new kind of strong unforgeability transformation which makes use of chameleon hash functions (Hash functions with a public-secret key pair and where a valid collision can be found using the private hash key). We can make use of this transformation as an alternative to the one suggested by [5]. The computation cost for this transformation is approximately the same compared to the one proposed by [5] as they have similar parameters and constructs. That is the probability of the forger breaking the scheme, is almost at the same level of the challenger solving the underlying hard problem(CDH Problem). We name this alternate scheme $PRS2_{SUF}$
Chameleon Hash Function: Originally coined and introduced by Krawczyk, was extensively used to implement blind signatures. It is associated with a public hash key and private hash key. One can compute the hash value using the public hash key. It is possible to find valid hash collisions only using the private hash key but cannot calculate the collision with only the chameleon hash and its public hash key. Let H_{ch} be a chameleon hash function with (m, s) as the input where m is the message and s is the randomness. It is easy to find a new pair $(\hat{m}, \hat{s})$ such that $H_{ch}(m, s) = H_{ch}(\hat{m}, \hat{s})$ with the knowledge of the private hash key. The construction for the chameleon hash function used is not an idealized one and hence it can use existing standard discrete log based constructions of chameleon hash functions [12].

Scheme Definitions:
KeyGen(1^k): Same as that in Scheme-1 where there are two pairs of secret,public key - $(g_2^{a_1}, g_1 = g^{a_1})$, $(g_2^{a_2}, g_{11} = g^{a_2})$ where $a_1, a_2 \in_R \mathbb{Z}_p$. Define a standard hash function $H_2 : \{0,1\}^* \rightarrow \mathbb{Z}_p$. The Chameleon Hash used in this scheme [12] is defined as follows: Using the generator g $\in \mathbb{G}$, set the element $g_3 = g^\beta$ where $\beta \in \mathbb{Z}_p$.
The private hash key is β and public hash key is (g_3, g).

$$\text{The chameleon hash } H_{ch}(m, s) = (g_3^m g^s).$$

Given a new $\hat{m} \neq m$, it is easy to find a hash collision using the private hash key β by calculating $\hat{s} = (m - \hat{m})\beta + s$. It can observed that
$H_{ch}(\hat{m}, \hat{s}) = g_3^{\hat{m}} g^{\hat{s}} = g_3^m g^s = H_{ch}(m, s)$.
g_3 is added to the common reference string for all users and β can be stored secretly for every user and is mainly used in the sign and re-sign algorithms. To note that the β used by each user(including proxy) is different.
Hence the common reference string, $< g, g_2, u_0, u_1 ... u_n, v_0, v_1 ... v_n, > \in \mathbb{G}$.
Sign(m, sk): On the input of the message to be signed and secret key of the user signing the message m, the signer to return the signature, performs the following computations:
Consider the primary secret of the signer (user A) to be $g_2^{a_1}$.

– Pick a random $\gamma, r \in \mathbb{Z}_p$ and compute g^{γ}.
– Consider $m_1 = H_1(g^{\gamma}) \in \{0,1\}^n$ as the message and perform the Waters' Signature on it as follows:
– Define $m_u^{(1)} = u_0 \prod_{i=1}^{n} u_i^{m_{1_i}}$.
where $m^{(1)} = (m_1^{(1)}, m_2^{(1)}, m_3^{(1)} \ldots m_n^{(1)})$ denotes the n-bit representation of $\mathrm{m}^{(1)}$.
– Compute $\sigma_1 = g_2^{a_1}.(m_u^{(1)})^r \in \mathbb{G}$.
– Compute $\sigma_2 = g^r \in \mathbb{G}$.
Unforgeability transformation
– Compute $m^{`} = H_2(m, \sigma_2) \in \mathbb{Z}_p$.
– Compute $s_1 = \gamma - m^{`}\beta$.
– Assign $\sigma_3 = s_1 \in \mathbb{Z}_p$
The signature $\sigma =< \sigma_1, \sigma_2, \sigma_3 >$

$$(\sigma_1, \sigma_2, \sigma_3) = \left(g_2^{a_1}.(m_u^{(1)})^r, g^r, s_1\right)$$

ReKey$(g_2^{a_1}, g_2^{b_1+b_2})$: In order to delegate the signing right from either A to B (or B to A as our scheme is bidirectional) the proxy will run an interactive protocol with A and B. The final re-signature key obtained by the proxy will be of the form

$$rk_{A\to B} = g_2^{b_1+b_2-a_1} \in \mathbb{G}$$

The Interactive protocol used to obtain the $rk_{A\to B}$ is the same as that defined in Section 3.
ReSign(pk_B, pk_A, m, σ): Proxy on receiving a signature $\sigma =< \sigma_1, \sigma_2, \sigma_3 >$ on the message m by user A associated with public key pk_A, re-signature key $rk_{A\to B}$ as input, performs the following to generate a signature on m for user B.
– First check whether the $Verify1(pk_A, m, \sigma)$ satisfies, if not abort reporting an invalid signature. Otherwise perform the following steps to generate the re-signature.
– Assign $\hat{\sigma}_2 = \sigma_2 = g^r \in \mathbb{G}$
– Let $\gamma_1, r_1 \in_R \mathbb{Z}_p$, compute and assign $\hat{\sigma}_3 = g^{r_1} \in \mathbb{G}$
– Consider $m_2 = H_1(g^{\gamma_1}) \in \{0,1\}^n$ as the message and perform the Waters' Signature on it as follows:
– Define $m_v^{(1)} = v_0 \prod_{i=1}^{n} v_i^{m_{2_i}}$.
where $m^{(1)} = (m_1^{(1)}, m_2^{(1)}, m_3^{(1)} \ldots m_n^{(1)})$ denotes the n-bit representation of $\mathrm{m}^{(1)}$.
Unforgeability transformation
– Compute $m^{``} = H_2(m, \hat{\sigma}_3) \in \mathbb{Z}_p$.
– Compute $s_2 = \gamma_1 - m^{``}\beta^{`}$.
– Assign $\hat{\sigma}_5 = s_2 \in \mathbb{Z}_p$
– Assign $\hat{\sigma}_4 = \sigma_3 = s_1 \in \mathbb{Z}_p$
Define $m_v^{(2)} = v_0 \prod_{i=1}^{n} v_i^{m_{2_i}}$
– Compute $\hat{\sigma}_1 = \sigma_1.rk_{A\to B}.(m_v^{(2)})^{r_1} \in \mathbb{G}$

$$= g_2^{a_1}.(m_u^{(1)})^r.g_2^{b_1+b_2-a_1}.(m_v^{(2)})^{r_1}$$
$$= g_2^{b_1+b_2}(m_u^{(1)})^r(m_v^{(2)})^{r_1}$$

The Re-signature $\hat{\sigma} = (\hat{\sigma}_1, \hat{\sigma}_2, \hat{\sigma}_3, \hat{\sigma}_4, \hat{\sigma}_5)$

$$= \left(g_2^{b_1+b_2}(m_u^{(1)})^r(m_v^{(2)})^{r_1}, g^r, g^{r_1}, s_1, s_2\right)$$

for one transformation of the signature is returned.
Verify1(m, σ, pk): For verifying the signature $\sigma =< \sigma_1, \sigma_2, \sigma_3 >$ on the message m by the user corresponding to public key g^{a_1}, any verifier can perform the following validity check.
– Compute $t = H_2(\sigma_2, m)$
– $m^{(1)} = H_1(g_3^t g^{\sigma_3}) \in \{0,1\}^n$
– $m_u^{(1)} = u_0 \prod_{i=1}^{n} u_i^{m_i^{(1)}}$.
– Verify whether the following equation satisfies

$$\hat{e}(\sigma_1, g) \stackrel{?}{=} \hat{e}(m_u^{(1)}, \sigma_2)\hat{e}(g_2, g_1)$$

if the above test holds, output *Valid* otherwise output *Invalid*.
Verify2$(m, \hat{\sigma}, pk)$: For verifying the re-signature $\hat{\sigma} =< \sigma_1, \sigma_2, \sigma_3, \sigma_4 >$ on the message m by the user corresponding to public key pk, any verifier can perform the following validity check on the transformed signature as follows.

– Compute $t = H_2(m, \hat{\sigma}_2) \in \{0,1\}^n$
– $m^{(1)} = H_1(g_3^t g^{\hat{\sigma}_4}) \in \{0,1\}^n$
– $t_1 = H_2(m, \hat{\sigma}_3) \in \{0,1\}^n$
– $m^{(2)} = H_1(g_3^{t_1} g^{\hat{\sigma}_5}) \in \{0,1\}^n$
– $m_u^{(1)} = u_0 \prod_{i=1}^{n} u_i^{m_i^{(1)}}$
– $m_v^{(2)} = v_0 \prod_{i=1}^{n} v_i^{m_i^{(2)}}$
– Verify whether the following equation satisfies

$$\hat{e}(\hat{\sigma}_1, g) \stackrel{?}{=} \hat{e}(m_u^{(1)}, \hat{\sigma}_2).\hat{e}(m_v^{(2)}, \hat{\sigma}_3).\hat{e}(g_2, g_1)\hat{e}(g_2, g_{11})$$

if the above test holds, output *Valid* otherwise output *Invalid*.

The Correctness of the Verify algorithm is similar to that given in PRS_{SUF}.

Proof of Security:

Theorem 2. *If there is an adversary, which can break the strongly unforgeable scheme $PRS2_{SUF}$ in polynomial time, by having q_s and q_{rs} queries to the sign and resign oracles and advantage (ϵ) with n as the size of the message, then the CDH problem can be broken with advantage $\epsilon^{``} \geq \epsilon/32q_s(q_s + q_{rs})(n + 1)$.*

In this theorem we give an overview of the proof of security for the strongly unforgeable scheme defined above. The proof of this theorem cannot use the proof of tight reduction that is defined in [7] due to the flaw pointed out in [8]. In [8] it has been proved that partitioning stratergy (such as [18]) has an inherent security loss of order 'q', where q is a polynomial. So we deviate from the proof of security in [7] and modify it similar to the proof in scheme 1.
Suppose there exists a (t,q_s, q_{rs}, ϵ) adversary that can break our strongly unforeable proxy re-signature scheme, then there is a challenger who can solve the computational Diffie-Hellman problem, i.e. when given a random tuple (g, g^a, g^b) then its output is g^{ab}. The proof is described in detail in Appendix B.

5.1 Efficiency Improvements of Schemes

Due to the use of a signature construct similar to that of Waters' [18] , the number of public parameters used in the scheme is quite high. Especially, the two n-vector group elements consume a enormous amount of memory which is not healthy considering the limited storage capacity available. As Patterson [10] had pointed out, we can use the techniques suggested by Naccache and Chatterjee-Sarkar who claimed that the number of parameters can be reduced by making a small modification in the manner we consider the n-vector group elements based on the n-bits of the hash function output. Instead of considering the message digest (hash function output) as a string of bits, it is taken as concatenation of t-bit integers. Hence number of group elements $\hat{n} = \frac{n}{t}$.
Consider the message m to consist of n bits. It is being split into t-bit integers and computation is modified accordingly,

$$u_0 \prod_{i=1}^{\hat{n}} u_i^{m_i} \text{ instead of } u_0 \prod_{i=1}^{n} u_i^{m_i}$$

This reduction in number of parameters also increase the efficiency by reducing the computations [10] performed during signing and resigning. But at the same time, there is a reduce in the success probability for the $\mathbb{C}$ to solve the underlying hard problem by the order of the number of signing and resigning queries during the training phase during the security game. According to [10] we can deduce that the probability is reduced by a factor of approximately $\frac{2^{q_s} 2^{q_{rs}}}{q_s q_{rs}}$. In order to compensate for the security loss, Chatterjee-Sarkar proposed an idea where the size of the group in which the CDH problem is hard. This will to an extent negate the effect of probability reduction due to q_s and q_{rs}.

6 Conclusion

We have presented in this paper, two strongly unforgeable proxy re-signature schemes, which can be proved secure in the standard model. We have made use of strong unforgeability transformation techniques to obtain the strongly unforgeable version of the signature scheme in order to make the resulting proxy re-signature scheme also as strongly unforgeable. However, there has been a trade off between efficiency and security, as we have by strengthening the security, reduced the efficiency due to the introduction of a large number of parameters. A few efficiency improvements have been suggested for the same. The future work for such strongly unforgeable, standard model (bi-directional/uni-directional) proxy re-signature schemes can be to propose and prove such schemes secure under the notion of adaptive corruption (for which the security model has been formalized [2] [6]) and also to increase its efficiency by reducing the number of parameters.

References

1. Jee Hea An, Yevgeniy Dodis, and Tal Rabin. On the security of joint signature and encryption. In *EUROCRYPT*, pages 83–107, 2002.
2. Giuseppe Ateniese and Susan Hohenberger. Proxy re-signatures: new definitions, algorithms, and applications. In *ACM Conference on Computer and Communications Security*, pages 310–319, 2005.
3. Mihir Bellare and Sarah Shoup. Two-tier signatures, strongly unforgeable signatures, and fiat-shamir without random oracles. Cryptology ePrint Archive, Report 2007/273, 2007.
4. Matt Blaze, Gerrit Bleumer, and Martin Strauss. Divertible protocols and atomic proxy cryptography. In *EUROCRYPT*, pages 127–144, 1998.
5. Dan Boneh, Emily Shen, and Brent Waters. Strongly unforgeable signatures based on computational diffie-hellman. In *Public Key Cryptography*, pages 229–240, 2006.
6. Sherman S. M. Chow and Raphael C.-W. Phan. Proxy re-signatures in the standard model. In *ISC*, pages 260–276, 2008.
7. Fuchun Guo, Yi Mu, and Willy Susilo. How to prove security of a signature with a tighter security reduction. In *ProvSec*, pages 90–103, 2009.
8. Dennis Hofheinz, Tibor Jager, and Edward Knapp. Waters signatures with optimal security reduction. In *Public Key Cryptography*, pages 66–83, 2012.
9. Benoît Libert and Damien Vergnaud. Multi-use unidirectional proxy re-signatures. In *ACM Conference on Computer and Communications Security*, pages 511–520, 2008.
10. Kenneth G. Paterson and Jacob C. N. Schuldt. Efficient identity-based signatures secure in the standard model. In *ACISP*, pages 207–222, 2006.
11. Jin Li Qiong Huang, Duncan S. Wong and Yi-Ming Zhao. Generic transformation from weakly to strongly unforgeable signatures. In *Journal of Computer Science and Technology*, volume 23, pages 240–252, 2007.
12. Adi Shamir and Yael Tauman. Improved online/offline signature schemes. In *CRYPTO*, pages 355–367, 2001.
13. Jun Shao, Zhenfu Cao, Licheng Wang, and Xiaohui Liang. Proxy re-signature schemes without random oracles. In *INDOCRYPT*, pages 197–209, 2007.
14. Jun Shao, Min Feng, Bin Zhu, Zhenfu Cao, and Peng Liu. The security model of unidirectional proxy re-signature with private re-signature key. In *ACISP*, pages 216–232, 2010.
15. Jun Shao, Guiyi Wei, Yun Ling, and Mande Xie. Unidirectional identity-based proxy re-signature. In *Communications (ICC), 2011 IEEE International Conference on*, pages 1–5, june 2011.
16. Ron Steinfeld, Josef Pieprzyk, and Huaxiong Wang. How to strengthen any weakly unforgeable signature into a strongly unforgeable signature. In *CT-RSA*, pages 357–371, 2007.
17. Isamu Teranishi, Takuro Oyama, and Wakaha Ogata. General conversion for obtaining strongly existentially unforgeable signatures. In *INDOCRYPT*, pages 191–205, 2006.
18. Brent Waters. Efficient identity-based encryption without random oracles. In *EUROCRYPT*, pages 114–127, 2005.

A Proof of Security for Scheme 1

Theorem 1 *If there is an adversary, which can break the strongly unforgeable scheme PRS_{SUF} in polynomial time, by having q_s and q_{rs} queries to the sign and resign oracles and advantage (ϵ) with n as the size of the message, then the CDH problem can be broken with advantage*

$$\epsilon^{''} \geq \epsilon/144q_s(q_s + q_{rs})(n+1)^2.$$

The security proof of the PRS_{SUF} can be divided into two parts. The first one is where we prove the security of the underlying signature scheme by simulating a weaker form of the signature with the sign and resign oracles. Thus we prove the scheme to be secure for any message that was not queried during the training phase. We are simulating the training phase for the weaker form of the re-signature(without transformation) where $m^{(1)}$ or $m^{(2)}$=m and prove for its existential unforgeability prior to applying the strong unforgeability transformation.
$\mathbb{C}$ is given the input of the hard problem, g^a, g^b and is required to find the solution g^{ab} from the forgery performed by the forger. Hence $\mathbb{C}$ embeds the hard problem instances while simulating the system to the adversary, which is described as follows.
Training Phase
Setup: Consider public key $g_1 = g^a$, $g_2 = g^b$ and therefore the solution to the hard problem is finding the secret key $g_2^{a_1} = g^{ab}$. The second secret key component $g_2^{a_2}$ does not involve in the hard problem solving and hence is generated in a similar fashion as in the KeyGen algorithm for each user.
Let the number of bits of the message be n. Let $l_n = 2(q_s + q_{rs})$ where q_s, q_{rs} are the number of queries to the sign/resign oracle, where $l_n(n+1) < p$ and let k_n be defined by $0 \leq k_n \leq n$. Let $x_0, x_1, ..., x_n \in \mathbb{Z}_{l_n}$ and similarly $y_0, y_1, ..., y_n \in \mathbb{Z}_p$.
$\mathrm{F(m)} = x_0 + \sum_{i \in U} x_i - l_n k_n$; $\mathrm{J(m)} = y_0 + \sum_{i \in U} y_i$

where U is set of all i from 1 to n, where $m_i = 1$. Let the parameters be
$u_0 = g_2^{x_0 - l_n k_n}.g^{y_0}$, $u_i = g_2^{x_i}.g^{y_i}$
Therefore, the hash function for Sign Oracle:
$m_u = u_0 \prod_{i=1}^{n} u_i^{m_i} = g_2^{F(m)} g^{J(m)}$
$l_m = 2q_{rs}$ where q_{rs} are the number of queries to the resign oracle, where $l_m(n+1) < p$ and let k_m be defined by $0 \leq k_m \leq n$. Let $z_0, z_1, ..., z_n \in \mathbb{Z}_{l_n}$ and similarly $w_0, w_1, ..., w_n \in \mathbb{Z}_p$.
K(m) $= z_0 + \sum_{i \in U^{`}} z_i - l_m k_m$; L(m) $= w_0 + \sum_{i \in U^{`}} w_i$
where $U^{`}$ is set of all i from 1 to n, where $m_i = 1$. Let the parameters be
$u_0 = g_2^{z_0 - l_m k_m}.g^{w_0}$, $u_i = g_2^{z_i}.g^{w_i}$
Therefore, the hash function for Resign Oracle:
$m_v = v_0 \prod_{i=1}^{n} v_i^{m_i} = g_2^{K(m)} g^{L(m)}$

Hence, it is evident from the above-mentioned steps that for any given message(m) $\mathbb{C}$ can calculate the respective hash function for the corresponding oracle.

- $O_{UKeyGen}$: When F queries for the key generation of user A, $\mathbb{C}$ does the following
  1. Selects an element $x_i, \hat{x}_i \in_R \mathbb{Z}_p$. Here $\hat{x}_i$ denotes the second secret key component.
  2. Computes public key $pk_A = (g^a g^{x_i}, g^{\hat{x}_i}) = (g^{a+x_i}, g^{\hat{x}_i})$ and sends pk_A to F. Note that the primary secret key sk_A of user A, is $(a + x_i)$ implicitly and $\mathbb{C}$ does not know primary sk_A which is used for signing.
- $O_{CKeyGen}$: When a query is made by F, $\mathbb{C}$ responds with $sk_i = (x_i, \hat{x_i})$ and $pk_i = (g^{x_i}, g^{\hat{x}_i})$ of a corrupted user where $x_i, \hat{x_i} \in_R \mathbb{Z}_p$.
- O_{ReKey}: On input with two public keys pk_i and pk_j $\mathbb{C}$ does the following, If both the users pk_i and pk_j (rekey between user i & j) are uncorrupted then $\mathbb{C}$ computes the rekey $g_2^{\hat{x_j} + \overline{x_j} - \overline{x_i}}$ where $\overline{x_j}$ and $\overline{x_i}$ are primary secret key components corresponding to pk_i and pk_j. The oracle returns $\perp$ and aborts if either one of the user corresponding to pk_i or pk_j is a corrupted user. The rekey $rk_{i \to j}$ is valid since by definition difference of the primary secret key component of the uncorrupted users $x_j - x_i$ when substituted with its values generated by $O_{UKeyGen}$ will be of the form $(a + \overline{x_j}) - (a + \overline{x_i})$ which is $\overline{x_j} - \overline{x_i}$. The second secret key component $\hat{x_j}$ will not be an issue, since it is a known value to $\mathbb{C}$.
- O_{Sign}: When F queries the sign oracle for message m to be signed by the primary secret key of the uncorrupted user corresponding to public key $pk_i = g^{a+x_i}$. If F(m) $\neq 0$ (F(m) was defined using the Setup), then return the following signature

$$\begin{aligned}
\sigma_1 &= g_1^{-J(m)/F(m)} (g_2^{F(m)} g^{J(m)})^r . g_2^{x_i} \\
&= g_2^{a+x_i} (g_2^{F(m)} g^{J(m)})^{-a/F(m)} (g^{J(m)} g_2^{F(m)})^r \\
&= g_2^{a+x_i} (g_2^{F(m)} g^{J(m)})^{r-a/F(m)} \\
&= g_2^{x_i} g^{ab} (g_2^{F(m)} g^{J(m)})^{r-a/F(m)} \\
\sigma_2 &= g_1^{-1/F(m)} g^r
\end{aligned}$$

$\sigma = (\sigma_1, \sigma_2)$

$$= \left(g_2^{x_i} g_1^{-J(m)/F(m)} \left(g_2^{F(m)} g^{J(m)} g_2^{x_i} \right)^r , g_1^{-1/F(m)} g^r \right)$$

Otherwise if F(m)=0 mod p, $\mathbb{C}$ aborts.

Correctness of Sign oracle for uncorrupted signature: This cooked up signature will satisfy the verification algorithm as follows:

$$\hat{e}(\sigma_1, g) \stackrel{?}{=} \hat{e}(u_0 \prod_{i=1}^{n} u_i^{m_i}, \sigma_2) \hat{e}(g_2, g_1)$$

where $\sigma_1 = g_2^{a+x_i} (g_2^{F(m)} g^{J(m)})^{r-a/F(m)}$

Right Hand Side:

$$\begin{aligned}
&= \hat{e}(u_0 \prod_{i=1}^{n} u_i^{m_i}, \sigma_2) . \hat{e}(g_2, g_1) \\
&= \hat{e}(g_2^{F(m)} g^{J(m)}, g^{r-a/F(m)}) . \hat{e}(g_2, g^{a+x_i}) \\
&= \hat{e}(g_2^{F(m)} g^{J(m)})^{r-a/F(m)}, g) . \hat{e}(g_2^{a+x_i}, g) \\
&= \hat{e}(g_2^{a+x_i} (g_2^{F(m)} g^{J(m)})^{r-a/F(m)}, g) \\
&= \hat{e}(\sigma_1, g).
\end{aligned}$$

Note: The fact that should be remembered while simulating the re-sign oracle is that for every uncorrupted user, the randomness in the signature is of the form $\hat{r}$=r-a/F(m) and for every corrupted user it is $\hat{r}$=r where $r \in_R \mathbb{Z}_p$

– O_{ReSign}: On input the signature σ on message m of the user corresponding to public key pk_i and the public key pk_j of the user to whom the signature is being transformed to, there are 3 possibilities for the input of this re-sign oracle. They are

1. Converting signature of an uncorrupted user to the signature of another uncorrupted user
2. Converting signature of a uncorrupted user to that of an corrupted user
3. Converting signature of an corrupted user to that of a uncorrupted user

$\mathbb{C}$ checks if Verify(m, σ, pk_i) is valid. If false $\mathbb{C}$ aborts, otherwise $\mathbb{C}$ does the following:
Consider $r_1 \in_R \mathbb{Z}_p$ to be the new randomness parameter used by the ReSign algorithm.
Case 1: If pk_i and pk_j are uncorrupted users, then $\mathbb{C}$ will call the $O_{ReKey}(pk_i, pk_j)$ to obtain $rk_{i\to j}$ and run the ReSign algorithm with new randomness $r_1 \in \mathbb{Z}_p$ and returns the re-signature to forger F.
Case 2: If pk_i corresponds to an uncorrupted user and pk_j to a corrupted user $\mathbb{C}$ is required to remove the hard problem instance g^{ab} while converting it to the corrupted user's signature. Hence the resulting ReSign must contain a g^{-ab} implicitly in order to cancel out the effect and thereby making it the signature of an corrupted user(whose primary secret key is $g_2^{x_j}$).
The input to the ReSign oracle for this case is as follows: $\sigma =< \sigma_1, \sigma_2 >$

$$\sigma_1 = g_1^{-J(m)/F(m)}(g_2^{F(m)}g^{J(m)})^r g_2^{x_i} g^{ab}(g_2^{F(m)}g^{J(m)})^{r-a/F(m)}$$
$$\sigma_2 = g_1^{-1/F(m)}g^r$$

the original randomness $\hat{r}$ = r - a/F(m).
The resignature for the corrupted user is calculated in the following manner:
Consider $r_1 \in \mathbb{Z}_p$

$$\begin{aligned}\hat{\sigma_1} &= \sigma_1 g_2^{x_j+\hat{x}_j} g_1^{L(m)/K(m)}(g_2^{K(m)}g^{L(m)})^{r_1}\\ &= g_2^{x_j+\hat{x}_j} g_1^{-J(m)/F(m)} g_1^{L(m)/K(m)}(g_2^{F(m)}g^{J(m)})^r(g_2^{K(m)}g^{L(m)})^{r_1}\\ &= g_2^{x_j+\hat{x}_j} g^{ab} g^{-ab}(g_2^{F(m)}g^{J(m)})^{r-a/F(m)}(g_2^{K(m)}g^{L(m)})^{r_1+a/K(m)}\\ &= g_2^{x_j+\hat{x}_j}(g_2^{F(m)}g^{J(m)})^{r-a/F(m)}(g_2^{K(m)}g^{L(m)})^{r_1+a/K(m)}\end{aligned}$$

Hence the new randomness $\hat{r_1}$ = r + a/K(m).

$$\hat{\sigma_2} = \sigma_2 = g_1^{-1/F(m)}g^r = g^{r-a/F(m)}$$
$$\hat{\sigma_3} = g_1^{1/K(m)}g^{r_1} = g^{r_1+a/K(m)}$$

$\mathbb{C}$ outputs $\hat{\sigma} = (\hat{\sigma_1}, \hat{\sigma_2}, \hat{\sigma_3})$ to the forger.

$$(\hat{\sigma_1}, \hat{\sigma_2}, \hat{\sigma_3}) = (g_2^{x_j+\hat{x}_j}\left(g_2^{F(m)}g^{J(m)}\right)^{r-a/F(m)}\left(g_2^{K(m)}g^{L(m)}\right)^{r_1+a/K(m)}, g^{r-a/F(m)}, g^{r_1+a/K(m)})$$

To be noted that the above computation is possible only if F(m) and K(m) $\neq$ 0 mod p. Otherwise, resign oracle returns $\perp$ and the game aborts.

Correctness of ReSign oracle for case 2: We show the correctness of the simulated $\hat{\sigma}$ as follows

$$\hat{e}(\hat{\sigma}_1, g) \stackrel{?}{=} \hat{e}(u_0\prod_{i=1}^{n} u_i^{m_i^{(1)}}, \hat{\sigma}_2).$$
$$e(v_0\prod_{i=1}^{n} v_i^{m_i^{(2)}}, \hat{\sigma}_3).\hat{e}(g_2, g_1).\hat{e}(g_2, g_{11})$$

Right Hand Side:

$$\begin{aligned}&= \hat{e}(u_0\prod_{i=1}^{n} u_i^{m_i^{(1)}}, \hat{\sigma}_2).\hat{e}(v_0\prod_{i=1}^{n} v_i^{m_i^{(2)}}, \hat{\sigma}_3).\hat{e}(g_2, g_1).\hat{e}(g_2, g_{11})\\ &= \hat{e}(g_2^{F(m)}g^{J(m)}, g^{\hat{r}})\hat{e}(g_2^{K(m)}g^{L(m)}, g^{\hat{r_1}})\hat{e}(g_2, g^{x_j+\hat{x}_j})\\ &= \hat{e}((g_2^{F(m)}g^{J(m)})^{\hat{r}}, g)\hat{e}((g_2^{K(m)}g^{L(m)})^{\hat{r_1}}, g)\hat{e}(g_2^{x_j+\hat{x}_j}, g)\end{aligned}$$

By Bilinearity property of the map e:

$$= \hat{e}(g_2^{x_j+\hat{x}_j}(g_2^{F(m)}g^{J(m)})^{\hat{r}}(g_2^{K(m)}g^{L(m)})^{\hat{r_1}}, g)$$
$$= \hat{e}(\hat{\sigma}_1, g).$$

Case 3: If pk_i corresponds to an corrupted user and pk_j to an uncorrupted user, then the $\mathbb{C}$ is required to induce the hard problem instance while converting it to the uncorrupted user's signature. Hence the resulting ReSign must contain a g^{ab} implicitly in order to induce the effect of the hard problem and thereby making it the signature of an uncorrupted user(whose primary secret key is $g_2^{x_j+a}$).
The input to the ReSign oracle for this case is as follows:

$$\sigma = \left(g_2^{x_i}.(u_0\prod_{i=1}^{n}u_i^{m_i})^r, g^r\right)$$
$$= \left(g_2^{x_i}(g_2^{F(m)}g^{J(m)}), g^r\right)$$

where the original randomness $\hat{r}$=r.
The resignature for the uncorrupted user is calculated in the following manner:
Consider $r_1 \in \mathbb{Z}_p$

$$\hat{\sigma}_1 = \sigma_1 g_2^{-x_i} g_1^{-L(m)/K(m)}(g_2^{K(m)}g^{L(m)})^{r_1}.g_2^{x_j+\hat{x}_j}$$
$$= g_2^{x_i} g_2^{-x_i} g_1^{-L(m)/K(m)}(g_2^{F(m)}g^{J(m)})^{r}(g_2^{K(m)}g^{L(m)})^{r_1}.g_2^{x_j+\hat{x}_j}$$
$$= g^{ab}(g_2^{F(m)}g^{J(m)})^{r}(g_2^{K(m)}g^{L(m)})^{r_1-a/K(m)}$$
$$= g_2^{x_j+\hat{x}_j} g^{ab}(g_2^{F(m)}g^{J(m)})^{r}(g_2^{K(m)}g^{L(m)})^{r_1-a/K(m)}$$

Hence the new randomness $\hat{r_1}$ = r - a/K(m).

$$\hat{\sigma}_2 = \sigma_2 = g^r$$
$$\hat{\sigma}_3 = g_1^{-1/K(m)}g^{r_1} = g^{r_1-a/K(m)}$$

$\mathbb{C}$ outputs $\hat{\sigma} = (\hat{\sigma}_1, \hat{\sigma}_2, \hat{\sigma}_3)$

$$= (g_2^{x_j+\hat{x}_j} g^{ab}\left(g_2^{F(m)}g^{J(m)}\right)^{r}$$
$$\left(g_2^{K(m)}g^{L(m)}\right)^{r_1-a/K(m)}, g^r, g^{r_1-a/K(m)})$$

to the forger. To be noted that the above computation can be performed only if K(m) $\neq$ 0 mod p. Otherwise, resign oracle returns $\perp$ and the game aborts.

Correctness of ReSign oracle for case 3: We show the correctness of the simulated $\hat{\sigma}$ as follows:

$$\hat{e}(\hat{\sigma}_1, g) \stackrel{?}{=} \hat{e}(u_0\prod_{i=1}^{n}u_i^{m_i^{(1)}}, \hat{\sigma}_2)\hat{e}(v_0\prod_{i=1}^{n}v_i^{m_i^{(2)}}, \hat{\sigma}_3)\hat{e}(g_2, g_1)\hat{e}(g_2, g_{11})$$

Right Hand Side:

$$= \hat{e}(u_0\prod_{i=1}^{n}u_i^{m_i^{(1)}}, \hat{\sigma}_2)\hat{e}(v_0\prod_{i=1}^{n}v_i^{m_i^{(2)}}, \hat{\sigma}_3)\hat{e}(g_2, g_1)\hat{e}(g_2, g_{11})$$
$$= \hat{e}((g_2^{F(m)}g^{J(m)}), g^{\hat{r}})\hat{e}((g_2^{K(m)}g^{L(m)}), g^{\hat{r_1}})\hat{e}(g_2, g^{a+x_j})\hat{e}(g_2, g^{\hat{x}_j})$$
$$= \hat{e}((g_2^{F(m)}g^{J(m)})^{\hat{r}}, g)\hat{e}((g_2^{K(m)}g^{L(m)})^{\hat{r_1}}, g)\hat{e}(g_2^{a+x_j+\hat{x}_j}, g)$$
$$= \hat{e}(g_2^{a+x_j+\hat{x}_j}(g_2^{F(m)}g^{J(m)})^{\hat{r}}((g_2^{K(m)}g^{L(m)})^{\hat{r_1}}, g)$$
$$= \hat{e}(\hat{\sigma}_1, g).$$

After the training phase, the forger F submits a **Forgery** : (m^*, pk^*, σ^*) for a message m^* corresponding to uncorrupted user's public key pk^*. If F is able to come up with a forgery for the uncorrupted user either on the signature or re-signature, then the Challenger $\mathbb{C}$ using the output of F, can find the solution for the CDH hard problem. Since the public key pk^* corresponds to an uncorrupted user, the forgery will always contain an instance of the hard problem solution in the signature.
The existential forgery on chosen message m^* is considered a valid forgery if it satisfies the following requirements:

1. None of the messages m_i among the q_s sign queries during the training phase has $F(m_i) \equiv 0 \; mod \; p$.
2. It is required that m^* is not among the Sign and ReSign queries in the training phase.

3. For a forgery of a re-signature, The condition $F(m^*) \equiv 0 \ mod \ p$ and $K(m^*) \equiv 0 \ mod \ p$ must satisfy in order for the $\mathbb{C}$ to compute the hard problem.

Signature of uncorrupted user (pk^*) is of the following form with the constraint that $F(m^*) \equiv 0 \ mod \ p$

$$(\sigma_1, \sigma_2) = \left(g_2^{x_i} g^{ab} \left(g_2^{F(m^*)} g^{J(m^*)} \right)^{r^*}, g^{r^*} \right)$$

Hence the solution to the CDH problem with respect to a *new* message m^*:

$$\begin{aligned} \sigma_1 &= g_2^{x_i} g^{ab} \left(g_2^{F(m^*)} g^{J(m^*)} \right)^{r^*} \\ &= g^{ab+bx_i+J(m^*)r^*} \end{aligned}$$

Therefore, $\frac{g^{ab+bx_i+J(m^*)r^*}}{(g^{bx_i}.g^{r^*)^{J(m^*)}}} = g^{ab}$

The forgery of the re-signature can be used to solve the CDH problem in a similar manner. Since the hard problem is induced accordingly depending on the nature of conversion between the uncorrupted and corrupted users.

The Re-Signature of uncorrupted user (pk^*) is of the following form with the constraint $F(m^*) \equiv 0 \ mod \ p$ and $K(m^*) \equiv 0 \ mod \ p$,

$$(\sigma_1, \sigma_2, \sigma_3) = (g_2^{x_j+\hat{x}_j} g^{ab} \left(g_2^{F(m^*)} g^{J(m^*)} \right)^{r^*} \left(g_2^{K(m^*)} g^{L(m^*)} \right)^{r_1^*}, g^{r^*}, g^{r_1^*})$$

Hence the solution to the CDH problem with respect to a *new* message m^*:

$$\sigma_1 = g_2^{x_j+\hat{x}_j} g^{ab} \left(g_2^{F(m^*)} g^{J(m^*)} \right)^{r^*} \left(g_2^{K(m^*)} g^{L(m^*)} \right)^{r_1^*} = g^{ab+bx_j+b\hat{x}_j+J(m^*)r^*+L(m^*)r_1^*}$$

Therefore, $\frac{g^{ab+bx_j+b\hat{x}_j+J(m^*)r^*+L(m^*)r_1^*}}{(g^{bx_j+b\hat{x}_j} g^{r^*)^{J(m^*)}} (g^{r_1^*)^{L(m^*)}})} = g^{ab}$

Hence the probability is calculated in accordance to the game not aborting in any query of the training phase and obeys the conditions stated in the forgery phase.

In the training phase, there are few instances in the simulation of the underlying signature when the game aborts. And the forgery is only calculated if the stated conditions are met. Hence, the event of abort:

$(\mathrm{F}(\mathrm{m}_i \equiv 0 \bmod \mathrm{p}) \bigvee (\mathrm{F}(\mathrm{m}_i \equiv 0 \bmod \mathrm{p} \bigwedge \mathrm{K}(\mathrm{m}_i \equiv 0 \bmod \mathrm{p}) \bigwedge (\mathrm{F}(\mathrm{m}^*) \not\equiv 0 \bmod \mathrm{p} \bigwedge \mathrm{K}(\mathrm{m}^*) \not\equiv 0 \bmod \mathrm{p})$

Hence the probability,

$$\Pr[\neg abort] \leq \Pr[F(m^*) \not\equiv 0 mod p \bigwedge K(m^*) \not\equiv 0 mod p] + \sum_{1}^{q_s+q_{rs}} \Pr[F(m^*) = 0 \bigwedge F(m_i) = 0] + \sum_{1}^{q_{rs}} \Pr[K(m^*) = 0 \bigwedge K(m_i) = 0]$$

Calculating the probability of the forger winning the game, in a similar fashion to that of [10] we obtain,

$$\epsilon' \geq \epsilon / 16 q_s (q_s + q_{rs})(n+1)^2$$

where q_s and q_{rs} is the total number of queries to the Sign and ReSign oracle. n is the number of bits of the message. In the notion of strong unforgeability, the forgery can be made on any message including those which have already been signed. We first simulated the weaker form of the underlying signature(without the transformation) which was proven existentially unforgeable, i.e. only a forgery on a message which has not been signed before is considered valid. Then the existential forgery of the signature is used to solve the CDH problem with a non-negligible probability. As stated earlier, the existential unforgeability result of the underlying signature scheme is transformed to a strongly unforgeable one using the stated transformation technique.

Now after applying the transformation [5] to both the signature and re-signature algorithms, where the message binds with the randomness and becomes a modified message input to the underlying existentially unforgeable re-signature scheme. As a result the re-signature becomes strongly unforgeable one and the security for the strong unforgeability is based on the security of the underlying existentially unforgeable signature scheme. (Proof can be referred from Theorem 1 of [5].)

The way, the transformation is setup over the underlying proxy re-signature against the following special kinds of adversaries,

1. When the forgery is of the form where $m^{(1)*} = m_i^{(1)}$ and $t_1^* = t_{1i}$ for $i \in 1, 2...q_s$ or $m^{(2)*} = m_i^{(2)}$ and $t_2^* = t_{2i}$ for $i \in 1, 2...q_{rs}$

2. When the forgery is of the form where $m^{(1)*} = m_i^{(1)}$ and $t_1^* \neq t_{1i}$ for $i \in 1, 2...q_s$ or $m^{(2)*} = m_i^{(2)}$ and $t_2^* \neq t_{2i}$ for $i \in 1, 2...q_{rs}$
3. When $m^{(1)*} \neq m_i^{(1)}$ for $i \in 1, 2...q_s$ or $m^{(2)*} \neq m_i^{(2)}$ for $i \in 1, 2...q_{rs}$.

It is observable that the strong unforgeability of the proxy re-signature follows the similar simulation by the $\mathbb{C}$ as described in [5]. It can hence be proved in a similar fashion (an extension of that in [5] that if the type 3 forger succeeds in his attempt it will break the underlying existentially unforgeable proxy re-signature which has been proved above. The type 1 forger is used to break the collision resistance of the hash function which is used to bind the message and randomness and the type 2 forger can be used to solve the discrete log problem in $\mathbb{G}$.
To note that the randomness of the original signature is bound with the message in the resign algorithm. This will retain the integrity of the re-signature and prevent it from being split up as independent components. This plays an important role to prevent the forgery on re-signatures.
After the application of the strong unforgeability transformation [5] to the signature and re-signature algorithms in PRS_{SUF} to get a strongly unforgeable one, the advantage of breaking the underlying existentially unforgeable re-signature scheme reduces to $1/3^{rd}$ since it is one of the three types of forgers according to Theorem 1 in [5]. Since we are applying to both the sign and re-sign algorithms, the advantage reduces to $1/9^{th}$. Hence the probability of solving the Computational Diffie-Hellman problem after applying the transformation is

$$\epsilon\text{“} \geq \epsilon/144q_s(q_s + q_{rs})(n+1)^2$$

B Proof of Security of Scheme 2

Theorem 2 *If there is an adversary, which can break the strongly unforgeable scheme $PRS2_{SUF}$ in polynomial time, by having q_s and q_{rs} queries to the sign and resign oracles and advantage (ϵ) with n as the size of the message, then the CDH problem can be broken with advantage $\epsilon\text{“} \geq \epsilon/32q_s(q_s + q_{rs})(n+1)^2$.*

In this theorem we give an overview of the proof of security for the strongly unforgeable scheme defined above. The proof of this theorem cannot use the proof of tight reduction that is defined in [7] due to the flaw pointed out in [8]. In [8] it has been proved that partitioning stratergy (such as [18]) has an inherent security loss of order 'q', where q is a polynomial. So we deviate from the proof of security in [7] and modify it similar to the proof in scheme 1.
We prove the security of this scheme in a single game with two types of adversaries.
- *Type 1:* The ability to forge on messages which were not queried during the training phase.
- *Type 2:* ability to forge on any message irrespective of being queried or not.

We will illustrate their formal definitions during the proof.
Informally, we are going to prove that, suppose there exists a (t,q_s, q_{rs}, ϵ) adversary that can break our strongly unforeable proxy re-signature scheme, then there is a challenger who can solve the computational Diffie-Hellman problem, i.e. when given a random tuple (g, g^a, g^b) then its output is g^{ab}. The initial training phase of the adversary is as follows:
Note: This scheme is proved secure against static corruption, where the corrupted entities are set prior to the beginning of the game.
The setup algorithm is similar to that in Theorem 1. The hard problem is inserted in the $g_2^{a_1}$ secret component of the uncorrupted user. There are some extra parameters which are to be introduced for the chameleon hash function, namely, $g_3 = g^{\beta}$, where $\beta \in \mathbb{Z}_p$.
Training Phase

The KeyGen and Rekey oracles for corrupted and uncorrupted users are taken in the same fashion as mentioned in Theorem-1. We assume $\hat{a}$ as the secondary secret component for both corrupted and uncorrupted users without loss of generality.
Setup of Waters hash for the Sign Oracle:
Let the number of bits of the message be n. Let $l_n = 2(q_s + q_{rs})$ where q_s, q_{rs} are the number of queries to the sign/resign oracle, where $l_n(n+1) < p$ and let k_n be defined by $0 \leq k_n \leq n$. Let $x_0, x_1, ..., x_n \in \mathbb{Z}_{l_n}$ and similarly $y_0, y_1, ..., y_n \in \mathbb{Z}_p$.
For setup for every i^{th} query is,
$F(M_i) = x_0 + \sum_{i \in U} x_i - l_n k_n$; $J(M_i) = y_0 + \sum_{i \in U} y_i$
where U is set of all i from 1 to n, where $M_i = 1$. Let the parameters be
$u_0 = g_2^{x_0 - l_n k_n}.g^{y_0}$, $u_i = g_2^{x_i}.g^{y_i}$
Therefore, the hash function for Sign Oracle:

$$M_i^u = u_0 \prod_{i \in M_i} u_i = g_2^{F(M_i)} g^{J(M_i)}$$

$\mathbb{O}_{UncorruptedSign}(m_i, pk_A) : \mathbb{C}$ chooses a $d_i \in \mathbb{Z}_p$ and computes a $M_i = H_1(m_i || d_i)$ which will be used as the M_i as defined in the setup above. Then a random r $\in \mathbb{Z}_p$, for a given m_i and computes the signature $(\sigma_1, \sigma_2, \sigma_3)$ as follows:

$$=(g_1^{-J(M_i)/F(M_i)}\left(g_2^{F(M_i)}g^{J(M_i)}\right)^r, g^r.g_1^{-1/F(M_i)}, d_i - m_i\beta)$$

There game aborts in the signing phase depending on the kind of d_i chosen. The abort scenario arises when $F(M_i) \equiv 0 \ mod \ p$

Setup of Waters hash for the ReSign Oracle:
$l_m = 2q_{rs}$ where q_{rs} are the number of queries to the resign oracle, where $l_m(n+1) < p$ and let k_m be defined by $0 \leq k_m \leq n$. Let $z_0, z_1, ..., z_n \in \mathbb{Z}_{l_n}$ and similarly $w_0, w_1, ..., w_n \in \mathbb{Z}_p$.
$K(M_i) = z_0 + \sum_{i \in U^{'}} z_i - l_m k_m$; $L(M_i) = w_0 + \sum_{i \in U^{'}} w_i$
where $U^{'}$ is set of all i from 1 to n, where $M_i = 1$. Let the parameters be
$u_0 = g_2^{z_0 - l_m k_m}.g^{w_0}$, $u_i = g_2^{z_i}.g^{w_i}$
Therefore, the hash function for Resign Oracle:

$$M_i^v = v_0 \prod_{i \in M_i} v_i = g_2^{K(M_i)}g^{L(M_i)}$$

$\mathbb{O}_{UnCorruptedReSign}(m_i, \sigma_i, pk_A, pk_B) : \mathbb{C}$ chooses a $d_i^{(2)} \in \mathbb{Z}_p$ and computes a $M_i = H_1(m_i||d_i)$ which will be used as the M_i as defined in the setup above. Then a random $r_1 \in \mathbb{Z}_p$ is chosen, for the given σ_i
First, the signature is verified using the Verify1 algorithm. If it does not satisfy, $\mathbb{C}$ returns $\perp$. Then we can induce the hard problem using the following construct $(\hat{\sigma}_1, \hat{\sigma}_2, \hat{\sigma}_3, \hat{\sigma}_4.\hat{\sigma}_5)$

$$=(g_2^{\hat{a}}g_1^{-L(M_i)/K(M_i)}\left(g_2^{F(M_i)}g^{J(M_i)}\right)^r\left(g_2^{K(M_i)}g^{L(M_i)}\right)^{r_1}, g^r, g^{r_1}.g_1^{-1/K(M_i)}, d_i^{(1)}, d_i^{(2)} - m_i\beta^{'})$$

where $\hat{a}$ is known by $\mathbb{C}$ and $d^{(1)}$ and $d^{(2)}$ is to show the notational difference between both the $d_i s$ chosen in the signing and resigning phase.
For an corrupted re-signature from an uncorrupted signature, a similar fashion as in Theorem 1 is adopted in order to where the hard problem is removed from the signature using the same technique of reversal of signs in the exponent and hence the resignature for corrupted user with secret key x_i would be of the form $(\hat{\sigma}_1, \hat{\sigma}_2, \hat{\sigma}_3, \hat{\sigma}_4.\hat{\sigma}_5)$

$$=(g_2^{\hat{a}+x_i}g_1^{L(M_i)/K(M_i)}(g_1^{-J(M_i)/F(M_i)}\left(g_2^{F(M_i)}g^{J(M_i)}\right)^r\left(g_2^{K(M_i)}g^{L(M_i)}\right)^{r_1}, g^r, g^{r_1}.g_1^{-1/K(M_i)}, d_i^{(1)}, d_i^{(2)} - m_i\beta^{'})$$

The abort scenario in the resign oracle arises when either $F(H_1(d_i^{(1)})) \equiv 0 \ mod \ p$ or $K(H_1(d_i^{(2)})) \equiv 0 \ mod \ p$

Forgery: The adversary outputs a valid signature $(\sigma_1^*, \sigma_2^*, \sigma_3^*)$ or a valid re-signature $(\sigma_1^*, \sigma_2^*, \sigma_3^*, \sigma_4^*, \sigma_5^*)$ for message m^*.
The forgery must always be done for an uncorrupted user, otherwise the hard problem cannot be solved and the game will abort.
Case 1: $m^* \neq m_i$ for i $\in \{m_1, ..., m_{q_s}/m_{q_{rs}}\}$ The forgery is on a new message. This is the same as that of existential forgeability.
With the security of the hash function we can conclude that

$$H_1(m_i, \sigma_{2_i}) \neq H_1(m^*, \sigma_2^*) \ \ \forall i \in \{1...q_s\}.$$

With the security of the chameleon hash we can conclude that

$$g_3^{H_1(m_i, \sigma_{2_i})}g^{\sigma_{3_i}} \neq g_3^{H_1(m^*, \sigma_2^*)}g^{\sigma_{3^*}}$$

The forgery must satisfy the fact that $F(M^*) = 0$, and is of the form:

$$(\sigma_1^*, \sigma_2^*, \sigma_3^*) = \left(g_2^a\left(u_0\prod_{i \in M^*} u_i\right)^{r^*}, g^{r^*}, s\right)$$

$\mathbb{C}$ solves the hard problem by performing the following computations,

$$\frac{\sigma_1^*}{(\sigma_2^*)^{J(M^*)}} = \frac{g_2^a\left(u_0\prod_{i \in M^*} u_i\right)^{r^*}}{g^{J(M^*)r^*}} = \frac{g_2^a\left(g_2^{F(M^*)}g^{J(M^*)}\right)^{r^*}}{g^{J(M^*)r^*}} = g_2^a = g^{ab}$$

When the forgery is for a re-signature, it must satisfy the fact that $F(M^*) = 0$ and $K(M^*) = 0$, and is of the form:

$$(\sigma_1^*, \sigma_2^*, \sigma_3^*, \sigma_4^*.\sigma_5^*) = (g_2^{a+\hat{a}}\left(u_0\prod_{i \in M^*} u_i\right)^{r^*}\left(v_0\prod_{i \in M^*} v_i\right)^{r_1^*}, g^{r^*}, g^{r_1^*}, s^*, s_1^*)$$

then the $\mathbb{C}$ solves the hard problem by performing the following computations,

$$\frac{\sigma_1^*}{(\sigma_2^*)^{J(M^*)}(\sigma_3^*)^{L(M^*)}g_2^{\hat{a}}} = \frac{g_2^{a+\hat{a}}\left(u_0\prod_{i\in M^*}u_i\right)^{r^*}\left(v_0\prod_{i\in M^*}v_i\right)^{r_1^*}}{g^{J(M^*)r^*}g^{L(M^*)r_1^*}g_2^{\hat{a}}} = \frac{g_2^a\left(g_2^{F(M^*)}g^{J(M^*)}\right)^{r^*}\left(g_2^{K(M^*)}g^{L(M^*)}\right)^{r_1^*}}{g^{J(M^*)r^*}g^{L(M^*)r_1^*}g_2^{\hat{a}}} \quad = g_2^a \quad = g^{ab}$$

Case 2: When $m^* = m_i$ it is evident that
$(\sigma_1^*, \sigma_2^*, \sigma_3^*) \neq (\sigma_{1_i}, \sigma_{2_i}, \sigma_{3_i})$ and $(\sigma_1^*, \sigma_2^*, \sigma_3^*, \sigma_4^*.\sigma_5^*) \neq (\sigma_{1_i}, \sigma_{2_i}, \sigma_{3_i}, \sigma_{4_i}, \sigma_{5_i})$, $\forall$ i = $\{m_1, ..., m_{q_s}/m_{q_{rs}}\}$.

In this type of forgery when $\sigma_2^* \neq \sigma_{2_i}$ or $\sigma_3^* \neq \sigma_{3_i}$ for the signature/re-signature, then the inequality will be similar to the above case.
When $\sigma_2^* = \sigma_{2_i}$ or $\sigma_3^* = \sigma_{3_i}$ for the signature/re-signature, we know $s^* \neq s_i$ or $s_1^* \neq s_{1_i}$. Hence we there will be an inequality of the chameleon hash and will lead to contradictory of the CDH assumption as seen before.
Thus the probability that the game will not abort is calculated in a similar fashion as in [10] and in our theorem 1 but with a slightly tighter bound as our strong unforgeability blends in a single game proof.

Pr[Case 1 or 2].Pr[Not abort in Forgery Phase] = $\dfrac{\epsilon}{32q_s(q_s+q_{rs})(n+1)^2}$

C Insecurities in direct application of strong unforgeability transformation

Now if we take Shao's Proxy Re-signature scheme and apply the transformation in [5], the underlying signature may become strongly unforgeable, but the resulting re-signature conversion will not be possible because of the re-randomization that happens while converting the signature to the re-signature. This can be illustrated as follows:

Table C.1 : Strong unforgeability transformation on Shao's Proxy Re-Signature [13]

Secret keys of A and B	g_2^a and g_2^b
Transformation [5]	$t = H_1(m, g^r)$, $\hat{m} = H_1(g^t h^s)$
Signature	$\sigma_1 = g_2^a.H_1(\hat{m})^r$, $\sigma_2 = g^r$,$\sigma_3 = s$
Rekey	rk = b/a
Re-Signature	$\hat{\sigma_1} = \sigma_1^{rk} = g_2^a.H_1(\hat{m})^{rb/a}$ with the new randomness components to be $\hat{\sigma_2} = g^{rb/a}$.

Note that the resulting re-signature will not satisfy the verify algorithm. This is because of binding of message m with the original randomness r. Thus re-randomization of the entire signature results in an invalid re-signature as $\hat{m}$ cannot be modified.
Due to this drawback, the transformation can be applied to the proxy re-signature scheme separately in the signing and the resigning phase. Instead of re-randomizing the entire signature, we introduce a new randomness as an extra component along with retaining the original randomness of the signature thereby making the proxy re-signature scheme transparent. Otherwise the following insecurity arises,

Table C.2: PRS_{SUF} Re-Signature without introducing the new randomness

Secret keys of A and B	g_2^a and g_2^b
Transformation [5]	$t = H_1(m, g^r)$, $\hat{m} = H_1(g^t h^s)$
Signature	$\sigma_1 = g_2^a.H_1(\hat{m})^r$, $\sigma_2 = g^r$, $\sigma_3 = s$
Re-Key	g_2^{b-a}
Re-Signature	$\hat{\sigma_1} = g_2^{b-a}.\sigma_1 = g_2^b.H_1(\hat{m})^r$, $\hat{\sigma_2} = \sigma_2 = g^r$, $\hat{\sigma_3} = \sigma_3 = s$
Insecurity (private proxy)	Forger passively obtains re-key using rk=$\hat{\sigma_1}/\sigma_1$=g_2^{b-a}

It is also evident from PRS_{SUF} that there is a dependency between both randomness components. This might protect the scheme from a strong unforgeability attack on the re-signature. Hence we bind the message and the old randomness with this new randomness using the same transformation in order to make sure that the resulting proxy re-signature scheme is also strongly unforgeable.

The problem of having just one secret key, public key pair will allow an adversary to generate re-signatures arbitrarily from signatures of the same user. Hence, in PRS_{SUF} using two key pairs the signature and re-signature differ by both a random component and a secret key component making it hard to forge a re-signature from a given signature.