Private Information Escrow Bound to Multiple Conditions

Abstract

We propose two variants of a protocol that provides users' private information escrow capability bound to multiple set of conditions in a federated single sign-on environment using trusted platform module (TPM) and secure processor technologies. The first variant assumes the existence of a trusted anonymity revocation manager, while the second variant does not. Cryptographic techniques, such as identity-based re-encryption and custodian-hiding encryption, are *applied* in our protocol. A performance analysis of our protocol is provided to show that our protocol achieves a better performance in comparison to the existing anonymous credential approach. This is especially true when a user interacts with many service providers in a session. The security properties provided by both variants of our protocol are discussed.

Keywords: privacy, conditional anonymity, anonymous credential, federated single sign-on

1 Introduction

Consider a scenario where a user u_a goes to an online shopping portal to do some purchases in a session, including a prescription medicine from a service provider X (SP_x), a few toys from an SP_y, and other goods from other SPs. For privacy reasons, u_a can be anonymous. However, for accountability purposes, some of u_a's personal identifiable information (PII) may have to be revealed when certain conditions are met - such as when the medicine u_a purchased was found to have serious previously-undetected side-effects. Therefore, the stores with whom u_a purchased the goods have to make sure that u_a's PII is correctly escrowed so that it can be revealed when needed. We call this the *conditional revelation of PII* (CRPI).

The existing anonymous credential system (ACS) - such as (Bangerter et al. 2004) - provide the CRPI capability through their PII-escrow operation. However, it suffers from a poor performance: if the user u_a makes purchases from r stores in a session, then r-number of PII-escrow operations have to be performed. This is because the PII-escrow operation binds a user's PIIs to a set of *Conditions-X* specific to an SP_x only. Due to the different nature of goods and services offered between SPs, it is highly likely that an SP_x will have a different set of conditions to another SP_y. As a PII-escrow operation requires many resources-intensive cryptographic operations (generation of commitments, encryptions, and execution of zero-knowledge proof protocol), having to perform such an operation multiple times in a session will easily result in a poor performance, thus reduced usability to the users. This problem is aggravated for users with limited-power devices.

The main contribution of this paper is a proposal for a protocol that allows a user to bind her escrowed PII to multiple set of conditions while only having to perform the resources-intensive PII-escrow operation ***once***. We achieve this by extending the existing ACS into a federated single sign-on (FSSO) environment, using trusted platform module (TPM) and secure processor technologies. The main advantage of our protocol is that, given the same scenario above, it achieves a significantly better performance as compared to using the existing approach (Bangerter et al. 2004). Section 7 provides a comparison of our protocol with the existing one.

Background Anonymous credential systems - such as (Bangerter et al. 2004) - provide many privacy enhancing features, including anonymous authentication (the ability to prove that a user is 'known' and certified by a trusted certificate authority without revealing the user's identity), selective disclosure of personal identifiable information (PII) in a certificate without revealing the whole certificate, conditional revelation of PII, and many others. This paper uses one of the most important capabilities of an ACS: CRPI.

Consider the scenario above. When u_a do not trust SP_x, or SP_y, or any other SPs in that portal, providing CRPI can be problematic: u_a might provide some PIIs to the SPs prior to purchasing goods, but this means that u_a is no longer anonymous. Besides, u_a can always provide bogus PIIs to the SPs. The PII-escrow operation provided by the existing ACS (Bangerter et al. 2004) provides a method to address this problem by escrowing u_a's PII to an anonymity revocation manager *ARM* as follows: u_a encrypts the required PII under a set of *Conditions-X* using the *ARM* public key and gives the ciphertext to SP_x. The value of *Conditions-X* is *part of the input* of the encryption process. Then, u_a and SP_x engage in a zero-knowledge proof (PK) protocol to convince SP_x that the provided ciphertext contains valid PII of u_a which have been *certified* by a trusted certificate authority (this also implies anonymous authentication) and that it can be decrypted by the *ARM*. To reveal the PII, the *ARM* has to be convinced that the *Conditions-X* are satisfied before decrypting the ciphertext. The decryption process requires the *ARM*'s private key as well as the same *Conditions-X* as used during the encryption process (see Appendix A.1 for more detailed explanation).

However, as mentioned earlier, such a PII-escrow operations requires the execution of many cryptographic operations which consume a substantial amount of computing resources. Having to perform such PII escrow with many SPs will easily result in a decreased performance and, subsequently, poor usability of the system to the user.

One method to improve the performance and usability of ACS is to use them in the context of the existing federated single sign-on (FSSO) systems, such as SAML 2.0 (OASIS 2005*a*) and WS-Federation (Lockhart et al. 2006). Such an approach has been proposed in (Suriadi, Foo & Josang 2008). In essence, an FSSO system allows a user to authenticate once to an identity provider (IdP) and then proceeds to access services from any service providers (SPs) in the federation (from the scenario above, the shopping portal can be the IdP, while SP_x and SP_y are the SPs). It is during this authentication stage that a user can perform the PII escrow operation (recall that the process of PII escrow also implies anonymous authentication). When an SP requires the user's authentication information, it will contact the IdP, to whom the user has been authenticated, to obtain an *assertion* containing the user's authentication information. If PII-escrow is used, the IdP, in the subsequent assertions issued to SPs, can simply include a statement stating that a set of user's PII have been escrowed and that they can be revealed when a certain set of *Conditions* are satisfied. By doing so, we can significantly reduce the number of resource-intensive cryptographic operations that PII-escrow operation required. Thus, in addition to improving performance, users also benefit from the *convenience* that FSSO provides, which results in an improved usability, while still retaining the privacy protections from the use of anonymous credential. An IdP can also play the role of an ARM, or they can be different entities.

Problem The PII escrow mechanism described earlier binds the encrypted PII to a set of *Conditions-SP-X* which is only valid between a user u_a and a service provider SP_x. Such one-to-one binding between encrypted PII and SP_x is problematic when applied in an FSSO environment: when A goes to another SP_y, a different set of *Conditions-SP-Y* may be used (*Conditions-SP-X* $\neq$ *Conditions-SP-Y*), and thus rendering the previously encrypted PII (bound to *Conditions-SP-Y*) unusable. Of course u_a and SP_x can perform the whole PII escrow operation again - including the commitments generation, encryption of the PII, and PK operations. However, this defeats the performance, convenience, and usability benefits gained from using FSSO.

Therefore, the *main problem* to address is how to bind a set of escrowed PII to multiple set of conditions (which may or may not have been determined prior to the start of a session) as efficient as possible in an environment where IdPs, SPs, and users are malicious. This issue is *not* addressed in (Suriadi, Foo & Josang 2008). We henceforth call this problem the *multiple conditions* problem.

Proposed Solution In this paper, we propose a protocol (with two variants) to address the *multiple conditions* problem. In the first approach, we propose the use of an identity-based encryption-proxy re-encryption scheme (IBEPRE), such as (Green & Ateniese 2007), in combination with the ACS (Bangerter et al. 2004). By re-encryption, we mean the transformation of a ciphertext C that was encrypted under a party X's public key to another ciphertext C' which a different party Y can decrypt using her private key without the entity who performs the re-encryption (the proxy) learning the value of the plaintext. IBEPRE allows arbitrary string (such as the *Conditions*) string to be used as a public encryption key. Therefore, conceptually, we should be able to re-encrypt a set of escrowed PII encrypted under a *Conditions-SP-X* to another ciphertext encrypted under *Conditions-SP-Y*. The main advantage of this approach is that it only requires minimal computations from the user's side, thus, meeting the efficiency and usability requirements.

However, the main drawback of using any identity-based encryption schemes (including IBEPRE) is that a single entity, known as the private key generator (PKG), can generate the private key for any given public key, and thus is able to decrypt any encrypted (or re-encrypted) messages (Baek et al. 2004). In our case, we can easily extend the role of an ARM to include that of a PKG. We henceforth call this problem the *trusted* ARM problem. When there is a trusted ARM, the solution just described can be applied. Otherwise, we need a different solution.

Therefore, we propose a second solution which combines the ACS with the custodian hiding encryption scheme (UCH) (Liu et al. 2005) to address the *multiple-conditions* and the *trusted* ARM problems. In this approach, we do not assume the existence of a single trusted ARM. Instead, such trust is distributed amongst n referees.

In both approaches, we take advantage of the the recent advancement in the Trusted Platform Module (TPM) technology (TCG 2007), along with secure processor technology that is already provided by most of the current processors, such as Intel Trusted Execution Technology (Intel 2007) (we henceforth call a TPM platform that uses secure processor technology as an *extended TPM* platform). Using an *extended TPM* platform, we can perform a secure execution of sensitive codes with *provable isolated execution* property: the ability to prove that a given output is the result of correct execution of a set of integrity-protected codes based on a verified given input (see (McCune et al. 2008) for details). As we shall see in section 4 and 5, the *provable isolated execution* property is very useful in providing a reasonable assurance of a user's 'correct behaviour' even when the user is assumed to be malicious.

In both variants of our protocol, we show that a better performance is achieved as compared to using the existing ACS approach alone, especially when a user needs to interact with many SPs in a session.

Contributions To summarize, the contributions of this paper are: (1) two variants of a protocol to enable secure PII-escrow bound to multiple conditions (PIEMC) in an FSSO environment (we henceforth call such a protocol FSSO-PIEMC): the first one assuming the existence of a trusted ARM, while the second one does not make such an assumption, and (2) a performance analysis of FSSO-PIEMC to show how our protocol achieves a better performance in comparison to the existing approach.

This paper is organized as follows: section 2 details the security requirements and the threat environment of an FSSO-PIEMC system. Section 3 describes the notations used and a brief description of the cryptographic schemes that are used in this paper (more detailed explanations on the cryptographic schemes used and TPM technologies are provided in Appendix A). Section 4 details the first variant of the FSSO-PIEMC protocol, assuming the existence of a trusted ARM. Section 5 details the second variant of the FSSO-PIEMC, without a trusted ARM. Section 6 provides an informal analysis of the security properties in both variants of the FSSO-PIEMC protocol. Section 7 analyses the performance of FSSO-PIEMC, and compares it to the existing approach. Conclusion and future work are provided in section 8.

2 Requirements and Threats

The main entities in FSSO-PIEMC are: users (U), IdPs, SPs, and ARM, and a set of referees R. The referees are only used in the second variant of the FSSO-PIEMC (without trusted ARM) to assess if a given set of *Conditions* is satisfied or not.

2.1 Requirements

The security requirements for an FSSO system that respects users' privacy have been detailed in (Bhargav-Spantzel et al. 2006).[1] Moreover, the requirements for a system that applies ACS while removing the need of a single trusted ARM has been proposed in (Suriadi, Foo & Smith 2008*b*,*a*). In this paper, we focus on the requirements needed for an FSSO-PIEMC system, therefore, we will extend some of the requirements from the cited references, and add a new one - *multiple conditions* - for the FSSO-PIEMC environment:

Main requirements:

- *Multiple conditions*: a user's escrowed PII must be able to be bound to multiple sets of conditions from various SPs *in a session.* These conditions may be determined prior, or after, the start of a session.
- *Zero-knowledge*: ARM, IdPs, referees, and unauthorized SPs must not be able to learn the value of the escrowed PII. Only *authorized* SPs can learn the values of the escrowed PII. By *authorized*, we mean the situation whereby the set of *Conditions* associated to a particular escrowed PII are satisfied. This requirement implies the *confidentiality, conditional release, and revocation* properties detailed in (Bhargav-Spantzel et al. 2006).
- *Enforceable conditions fulfillment*: this property can be broken down into two types:
  - *Direct Enforcement of Conditions* means that revocation of the escrowed PII is dependent on actions which are **directly** related to conditions fulfillment (such as in in e-cash applications (Davida et al. 1997))
  - *Indirect Enforcement of Conditions* means that the fulfillment of *Conditions* has to be performed 'manually', however, once such fulfillment of conditions is confirmed, certain actions can be performed (such as decrypting a ciphertext) that will result in the revelation of the escrowed PII.

  This requirement is used in (Suriadi, Foo & Smith 2008*b*), and it is similar to the *privacy policy, obligations, restrictions, and enforcement* properties detailed in (Bhargav-Spantzel et al. 2006).
- *Authenticated PII escrow*: while the value of the escrowed PII are not known (anonymous), the recipient of the escrowed PII (such as an IdP) has to be convinced that when the escrowed PII are recovered, they will reveal the correct information. This property is extended from the *authenticated user* property used in (Suriadi, Foo & Smith 2008*b*). This property encompasses the *anonymity, verifiability, integrity, and confidentiality* properties in (Bhargav-Spantzel et al. 2006)
- *Conditions-Abuse resistant*: the FSSO-PIEMC should be designed such that it is secure from malicious IdPs, SPs, and referees who may try to 'down grade' the security of the system by binding a set of escrowed PII with easy-to-fulfill conditions. Similarly, FSSO-PIEMC should also resist an attempt by users to bind a set of escrowed PII with a set of conditions which may be hard or impossible to satisfied. This requirement is extended from the more generic *Abuse resistant* property defined in (Suriadi, Foo & Smith 2008*b*).

Desirable requirements:

- *Minimum on-line computations*: while online cryptographic-related computations may not be avoided altogether, we need to keep them to a minimum. Therefore, we need to (1) reduce the number of computations required, and (2) offload as many computations as possible to off-line processing.
- *User-centric*: the revocation of a user's escrowed PII can only be successfully performed with the knowledge (and possibly direct participation) of the user. The *user-centric* property is beneficial if users are honest. If users are aware that a certain PII of theirs are being revoked in an *unauthorized* manner, they can immediately take corrective actions. However, this property may not be desirable in some cases, such as terrorism activities detection and prevention. This requirement encompasses the *user-in-the-middle, user-consent, and user notification* properties in (Bhargav-Spantzel et al. 2006), and it is used in (Suriadi, Foo & Smith 2008*b*)
- *(non)-Forward secrecy*: The nature of FSSO means that, even with the use of anonymous credential, IdP will be able to track an anonymous user's activities *within* a session. However, once a user's PII is revoked, we may or may not want to link the revoked PII to a set of past sessions. If the *forward secrecy* property is not supported, then sessions will be linkable. *Linkable sessions* might be more practical (especially for security investigation purpose), however, *unlinkable sessions* property provides a better privacy for users. This property is called the *events linkability* property in (Suriadi, Foo & Smith 2008*b*), and it is similar to the *unlinkability* property detailed in (Bhargav-Spantzel et al. 2006).

2.2 Threats

In this paper, we consider threats that can be executed by either the users, IdPs, SPs, ARM, and referees. Other threats which have been addressed in the existing FSSO standards, such as replay of assertions attack - see (OASIS 2005*b*), are not considered. Therefore, the threat environment for the FSSO-PIEMC protocol is as follows:

- Malicious users who may provide false PII (for escrow) and may attempt to cause unsuccessful revocation of the escrowed PII even when *Conditions* are satisfied.
- Malicious IdPs and SPs who will attempt to reveal the escrowed PII in an un-authorized manner. However, as is common in an FSSO model, SPs trust IdPs.
- Honest referees ($R_h \subset R$), with a small subset of dishonest referees ($R_{dh} \subset R$) who may also

[1] In (Bhargav-Spantzel et al. 2006), they call such system the universal user-centric system

attempt to reveal the escrowed PII in an unauthorized manner.

- Honest ARM for FSSO-PIEMC with trusted ARM (section 4). For the FSSO-PIEMCE without trusted ARM (section 5), no ARM is involved.
- Collusion: possible between IdPs, SPs, and R_{dh}. However, collusion between U and IdPs, SPs, ARM, or R_{dh} is unlikely due to conflicting interests: U wants to protect the PII from being revealed, while IdPs, SPs, ARM, and R_{dh} have the exact opposite interest

3 Notations and Concepts

$m_a, ...m_e$ are plaintext PII. An encryption of a PII m_a using an encryption *scheme* under an entity i's public encryption key and a label $Label$ is denoted as: $Cipher^{K^i_{pub-scheme},Label}_{scheme-m_a} = Enc_{scheme}(m_a; Label; K^i_{pub-scheme})$. Only the entity i who has the corresponding private key $K^i_{priv-scheme}$ can decrypt and recover $m_a = Dec_{scheme}(Cipher^{K^i_{pub-scheme},Label}_{scheme-m_a}; K^i_{priv-scheme})$.

A signature of an entity i over a message m_a can only be produced by using i's signing key (which is private): $S^{K^i_{sign}}_{m_a} = Sign(m_a; K^i_{sign})$. This signature can be verified by anybody using the signature verification key of i (which is public): $VerifySign(S^{K^i_{sign}}_{m_a}; m_a; K^i_{verify}) = 1$ (valid) or 0 (invalid).

A commitment of PII m_a is generated using a $Commit$ algorithm, with a random value r: $c_{m_a} = Commit(m_a, r)$. A commitment should be *hiding* (does not reveal any computational information on m_a) and *binding* (computationally infeasible to find another m_j and r' as inputs to the same $Commit$ algorithm that gives a value $c_{m_j} = c_{m_a}$).

A zero knowledge proof interactive protocol (PK) executed between a Prover and a Verifier is denoted as follows: $PK\{(m_a)$: $F(m_a, m_b...m_e) = 1\}$. The PII on the left of the colon m_a is the PII that a Prover needs to prove the knowledge of such that the statements on the right-side, $F(m_a, m_b...m_e) = 1$, is correct. A verifier will not learn the value of the PII on the left hand side of the colon, while other parameters are known. A PK may involve one or more message exchange(s). At the end of a PK execution, a verifier will (or will not) be convinced that the prover has the knowledge of m_a without the verifier learning its value.

An ACS - such as (Bangerter et al. 2004) - uses many PK to provide its privacy-enhancing features, such as CRPI. A certificate $Cert$ in an ACS is a signature of a certificate issuer $CertIssuer$ over a collection of PII. A $Cert$ is *private* to the user. To provide CRPI, a user first commits the PII to escrow (say m_a contained in a certificate $Cert$), then encrypts it using the verifiable encryption (VE) scheme (Camenisch & Shoup 2003*a*). Then, a PK is executed between the user and the recipient of the ciphertext to prove that the ciphertext correctly hides m_a as certified in $Cert_a$. More details on ACS are provided in Appendix A.1.

In IBEPRE schemes - such as (Chu & Tzeng 2007, Green & Ateniese 2007), public key is just a label, known as *id*. We denote an IBEPRE encryption of a message m_a under a label id_1 as $Cipher^{id_1}_{IBEPRE-m_a}$. To decrypt, the private key sk_{id_1} has to be extracted from a PKG (who has the master secret key `msk`). A re-encryption key $rk_{id_1 \rightarrow id_2}$ (which can be generated if one knows sk_{id_1}, id_1, and id_2) is needed to re-encrypt $Cipher^{id_1}_{IBEPRE-m_a}$ into $Cipher^{id_2}_{IBEPRE-m_a}$. The entity performing such a re-encryption does not learn the value of m_a. See Appendix A.2 for a more detailed description of IBEPRE scheme.

A universal custodian-hiding - $UCH(k, t, n)$ - encryption scheme (Liu et al. 2005) allows one to encrypt a message m_a such that any k members of a subgroup T (of t members) have to work together to decrypt it. The subgroup T is formed spontaneously by the encryptor of m_a from the main group N consisting of n members ($1 \leq k \leq t \leq n$). The identities of the members of T are hidden from the recipient of such an encryption. An encryptor can form the subgroup T consisting of different members with each encryption. A $UCH(k, t, n)$ encryption of m_a will result in t well-formed ciphertext pieces encrypted using the corresponding public keys of members of T, and $n - t$ random values chosen from specific domains such that they are indistinguishable from the well-formed ones: $Cipher^{K^{s_i}_{pub-UCH},Conditions}_{UCH(k,t,n)-m_a} + random^{r_i}$ (for $i = 1...n$, $s \in T$, and $r \in [1, n] \setminus T$). See Appendix A.3 for a more detailed description.

To verify that a user is using a genuine TPM in the most privacy-respecting manner, a Direct Anonymous Attestation (DAA) protocol, as proposed in (Brickell et al. 2004), must be used. A successful execution of DAA protocol convinces the verifier that it is interacting with a genuine TPM platform without learning the 'identity' of the platform. Instead, a pair of per-session Attestation Identity Key (AIK) is generated which can be used by the TPM as its authenticated signing key to sign TPM-generated messages - such as Platform Configuration Register (PCR) value - for that session only (PCR values contain the hash values of modules loaded in a secure execution area). Therefore, interactions with the same TPM over multiple sessions are unlinkable, hence better privacy.

The *provable isolated execution* property provided by an *extended TPM* plaform allows one to prove that a given output is the result of correct execution of a set of integrity-protected codes based on a given input which can also be verified. The generation of such proofs only require a simple signature of the TPM PCR value, input, output, and other parameters. Appendix A.4 provides more explanation of TPM and its related technologies. Readers who are interested in the details should consult (McCune et al. 2008).

4 FSSO-PIEMC with a trusted ARM

In this section, we detail the FSSO-PIEMC protocol which combines the ACS (Bangerter et al. 2004) with an IBEPRE scheme, using *extended TPM* technology. Our FSSO-PIEMC protocol is designed such that any IBEPRE schemes respecting the definition provided in (Green & Ateniese 2007) can be used. Readers who are not familiar with the mentioned cryptographic schemes and technologies should refer to the explanations provided in Appendix A.1, A.2, and A.4.

Overview We divide FSSO-PIEMC into few stages: setup, PII escrow, key escrow, multiple binding, and revocation. The setup only has to be performed once. The PII escrow, and key escrow stages only have to be performed once per session. The multiple binding stage can be performed multiple times in a session as needed. The revocation stage is only performed when an SP believes that a set of conditions is satisfied, thus needing the user PII to be re-

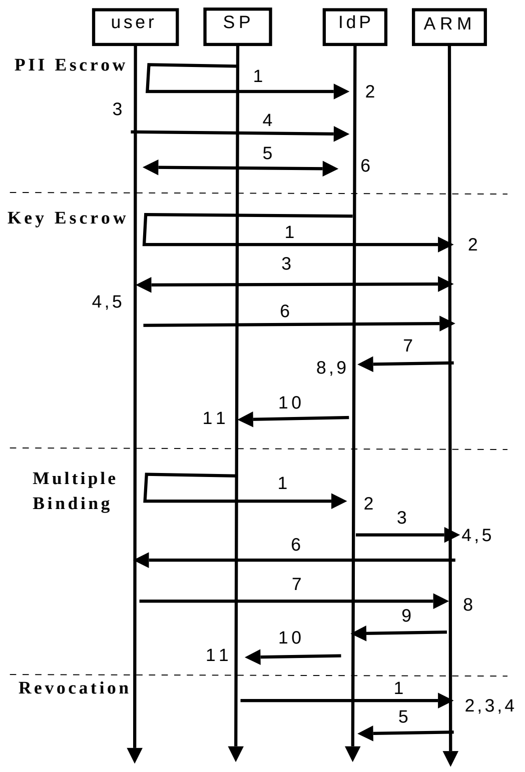

Figure 1: FSSO-PIEMC with trusted ARM

vealed. We assume the existence of a trusted *ARM* who will also perform the role of a PKG. This assumption is removed later in our second variant of the FSSO-PIEMC protocol detailed in section 5.

Assume we need to escrow *d*-number of a user's PII. We leave the original PII-escrow operation the same, except that in FSSO-PIEMC, instead of encrypting the PII to an *ARM*, we use a one-time user-generated keys for the verifiable encryption (VE) scheme (Camenisch & Shoup 2003*a*). The encryptions of the *d* PII are given to IdP for escrow. Then, the private portion of the VE one-time key is escrowed as follows: the user's *extended TPM* platform will perform a *provable isolated execution* of `Module1` - given the public key portion of the generated one-time VE keys, perform an IBEPRE encryption of the corresponding VE one-time private key under a condition string *Conditions* - see Table 1. Then, the encrypted key is sent to the *ARM* for escrow, along with the proof to show that the ciphertext is the result of a correct execution of `Module1`. When the escrowed PII needs to be bound to a separate *Conditions*2, the IdP will request the *ARM* to perform an IBEPRE of the IBEPRE-encrypted VE private key.

We require that a condition string *Conditions* to contain information about the identity of the SP to whom the conditions apply, a unique one-time random value, and a list of conditions that must be satisfied before the associated encrypted message can be recovered. The details of our FSSO-PIEMC protocol assuming a trusted *ARM* are as follows (see Figure 1):

Setup A user u_a obtains a certificate *Cert* containing PII $m_a...m_i$ from a certificate issuer *CertIssuer*. The PII certified by *CertIssuer* is accepted by the IdPs and SPs in the federation. To verify the issued certificate, the *CertIssuer*'s signature verification key $K^{CertIssuer}_{verify}$ is used.

PII escrow An FSSO session is started when a user u_a needs to access services from a service provider $SP1$ who in turn requires the IdP to escrow some of u_a's PII under a set of conditions *Conditions-SP1* (which can be freshly negotiated per session, or pre-agreed beforehand). A start of a session triggers the start of a PII escrow stage. While the existing implementation of PII escrow in (Bangerter et al. 2004) binds a set of conditions to the encryptions of those PII directly, in FSSO-PIEMC, such binding will only be reflected during the key escrow stage. At the PII escrow stage, we can use a generic condition string stating that 'decryption of these PII should only be performed pending a successful recovery of the associated decryption key that is escrowed in the following key-escrow stage'. We denote this condition as *GenCond.* The IdP will not be able to decrypt these escrowed PII because it does not have the decryption key.

1. $SP1$ generates a signed request for PII $m_a...m_c$ to be escrowed to the IdP. This request message includes the standard information dictated by the FSSO protocol used in the federation, as well as *Conditions-SP1.* This request message is redirected through u_a to the IdP.
2. The IdP verifies the request from $SP1$. If valid, it will contact the user to start the PII escrow operation.
3. User u_a generates:
   (a) one-time VE encryption key: K^u_{pub-VE}, $K^u_{priv-VE}$.
   (b) the commitments $c_a...c_c$ for PII $m_a...m_c$ respectively
   (c) encryptions of PII $m_a...m_c$, e.g. $Cipher^{K^u_{pub-VE}}_{VE-m_a} = Enc(m_a, GenCond; K^u_{pub-VE})$
4. u_a sends K^u_{pub-VE}, $Cipher^{K^u_{pub-VE}}_{VE-m_a}$... $Cipher^{K^u_{pub-VE}}_{VE-m_c}$ to the IdP
5. u_a and IdP engage in a PK to prove that the commitments $c_a...c_c$ hide PII $m_a...m_c$ which have been certified in *Cert* issued by *CertIssuer*. This PK also proves that $Cipher^{K^u_{pub-VE}}_{VE-m_a}$... $Cipher^{K^u_{pub-VE}}_{VE-m_c}$ are the correct encryptions of PII $m_a...m_c$ w.r.t K^u_{pub-VE}. The value of $c_a...c_c$ are given to the IdP as part of the PK procedure.

$$PK\{(Cert, m_a...m_c):$$
$$c_a = Commit(m_a, r) \wedge ...c_c = Commit(m_c, r) \quad (1)$$
$$\wedge VerifySign(m_a, .., m_i; K^{CertIssuer}_{verify}) = 1 \quad (2)$$
$$\wedge Cipher^{K^u_{pub-VE}}_{VE-m_a} = Enc_{VE}(m_a; GenCond; K^u_{pub-VE}) \wedge ... \quad (3)$$
$$\wedge Cipher^{K^u_{pub-VE}}_{VE-m_c} = Enc_{VE}(m_c; GenCond; K^u_{pub-VE})\} \quad (4)$$

6. After a successful execution of the above PK, the IdP stores K^u_{pub-VE} and $Cipher^{K^u_{pub-VE}}_{VE-m_a}...Cipher^{K^u_{pub-VE}}_{VE-m_a}$,and *Conditions-SP1.*

We cannot simply substitute the execution of the above PK protocol with the use of an *extended TPM* platform. This is because at this stage, the IdP does not have any data that can be used as a source of a *valid input* to a TPM module (garbage-in garbage-out problem). In our protocol, since it is the user who generated and gave K^u_{pub-VE} to the IdP, the IdP has to verify that the given K^u_{pub-VE} is correct in relation to the encrypted PII. The execution of the PK during the PII escrow stage above provides such a verification of K^u_{pub-VE} (that it is the correct encryption key used to encrypt $m_a...m_c$). Therefore, from this point onwards, we can use K^u_{pub-VE} and other publicly known value as inputs to a module to be executed by an *extended TPM.*

Key escrow The public key parameters `params-IBEPRE` for the IBEPRE scheme used is known to all participants in the federation.

1. The IdP signs *Conditions-SP1* and sends $S^{K^{ARM}_{sign}}_{\textit{Conditions-SP1}}$ + *Conditions-SP1* in a redirection message through the user to a chosen ARM.

2. The ARM verifies $S^{K^{ARM}_{sign}}_{\textit{Conditions-SP1}}$. If valid, it stores *Conditions-SP1* and continues. Otherwise, halt.

3. u_a and ARM engage in a DAA protocol to verify that the user is using a valid TPM platform and to generate a pair of AIK keys, denoted as $K^{TPM-u_a}_{verify-AIK}$, $K^{TPM-u_a}_{sign-AIK}$

   (a) As AIK should only be used to sign messages generated internally by a TPM, u_a's TPM should also generate a one-time signing key to sign messages from u_a (but not TPM-internally generated messages): $K^{u_a}_{verify}$, $K^{u_a}_{sign}$. The user's TPM can send $K^{u_a}_{verify} + S^{K^{TPM-u_a}_{sign-AIK}}_{K^{u_a}_{verify}}$ to the IdP for verification.

4. u_a runs `Module1` on her *extended TPM* platform to perform an IBEPRE encryption of $K^u_{priv-VE}$ under *Conditions-SP1* - see Table 1 for the details of `Module1`. This module will generate an output: $Cipher^{Conditions-SP1}_{IBEPRE-K^u_{priv-VE}}$

5. The platform should generate the proof of correct execution of `Module1`. This proof would contain information on the *extended TPM*'s PCR values (before and after execution) calculated from the value of `Module1`, the inputs, output, and other necessary information. This proof is signed using $K^{TPM-u_a}_{sign-AIK}$. See (McCune et al. 2008) for details of how such proof is generated.

6. The TPM proof + $Cipher^{Conditions-SP1}_{IBEPRE-K^u_{priv-VE}}$ are sent to the ARM.

7. The ARM verifies the TPM proof. If valid, the ARM stores *Conditions-SP1* + $Cipher^{Conditions-SP1}_{IBEPRE-K^u_{priv-VE}}$, and sends a signed $Cipher^{Conditions-SP1}_{IBEPRE-K^u_{priv-VE}}$ to the IdP

8. The IdP verifies the ARM signature on $Cipher^{Conditions-SP1}_{IBEPRE-K^u_{priv-VE}}$. If valid, it generates a one-time pseudonym $pseudo_a$ to identify the user for that particular FSSO session only

9. The IdP also stores $pseudo_a$, and links it with $Cipher^{Conditions-SP1}_{IBEPRE-K^u_{priv-VE}}$, K^u_{pub-VE}, *Conditions-SP1*, and $Cipher^{K^u_{pub-VE}}_{VE-m_a}...Cipher^{K^u_{pub-VE}}_{VE-m_c}$.

10. The IdP then sends a signed response message back to $SP1$. Included in the response are $pseudo_a$, $Cipher^{Conditions-SP1}_{IBEPRE-K^u_{priv-VE}}$, *Conditions-SP1*, $Cipher^{K^u_{pub-VE}}_{VE-m_a}...Cipher^{K^u_{pub-VE}}_{VE-m_c}$, the identity of the chosen ARM, and other messages specified by the FSSO protocol used.

11. $SP1$ verifies the response message from the IdP. If valid, $SP1$ now has the necessary data such that when *Conditions-SP1* are satisfied, $m_a...m_c$ can be recovered with the help of the ARM.

We might be able to remove the key escrow stage and have the user's *extended TPM* platform directly perform a *provable isolated execution* of IBEPRE encryptions of the user's PII hidden in those commitments which *have been proven* to hide the correct certified PII (from the execution of the PK during the PII-escrow stage). However, this approach is not scalable: assuming that in a session, d-number of PII are escrowed, when these escrowed PII need to be bound to another conditions, we now need to do d IBEPRE re-encryptions. With key escrow, we only need to perform the re-encryption of the VE private key, thus it is more efficient.

Another advantage of having the key escrow stage is that we can improve efficiency by performing the PII escrow stage even *before* the start of a session (only step 3 to 6 of the PII escrow, moving step 1 and 2 of the PII escrow stage to the key escrow stage). A user can escrow some commonly needed PII which are most likely to be escrowed before a session is started. This effectively 'saves up' a collection of escrowed PII to be used for future sessions. When a session is started between a user and an SP, the user only needs to tell the IdP which one-time K^u_{pub-VE} to use, and they can go straight to the key escrow stage.

Input	K^u_{pub-VE}, *Conditions*, `params-IBEPRE`
Process	P1.1. Retrieve $K^u_{priv-VE}$ P1.2. Verify that $K^u_{priv-VE}$ is the correct private key for the input value K^u_{pub-VE}
	P1.3. If P1.2 returns true, generate $Cipher^{Conditions}_{IBEPRE-K^u_{priv-VE}}$. Otherwise, return an error
Output	$Cipher^{Conditions}_{IBEPRE-K^u_{priv-VE}}$ or error

Table 1: `Module1` -IBEPRE encryption of VE private key

Multiple Conditions Binding This stage is started when the user u_a goes to another $SP2$ who also needs the escrowed PII but this time bound to a different set of conditions. Depending on the implementation, u_a and $SP2$ can negotiate the *Conditions-SP2* under which the escrowed PII can be revoked, or, they could use the pre-agreed *Conditions-SP2.*

1. $SP2$ generates a signed request for u_a's escrowed PII $m_a...m_c$ to be bound to *Conditions-SP2.* This request message include the standard request message dictated according to the respective FSSO protocol used in the federation, as well as *Conditions-SP2.*

2. The IdP verifies the request from $SP2$. From this request, the IdP will also detect that it has an open authenticated session with a user known as $pseudo_a$.[2]

3. The IdP retrieves *Conditions-SP1* associated with $pseudo_a$, and sends a signed re-encryption request to the ARM by sending *Conditions-SP1* and *Conditions-SP2'*.

4. The ARM verifies the request, and checks if it has the same *Conditions-SP1*. If not, stops.

5. To verify that the IdP has not given an invalid *Conditions-SP2'* to the ARM, the ARM prepares a message *multiple-bind* = *Conditions-SP1* + *Conditions-SP2'*, and generates a signature over *multiple-bind*.

6. The ARM sends $S^{K^{ARM}_{sign}}_{multiple-bind}$ to u_a

7. The user u_a, who knows *Conditions-SP1* and *Conditions-SP2* verifies $S^{K^{ARM}_{sign}}_{multiple-bind}$, and if valid, sends a signed 'OK' message using $K^{u_a}_{verify}$ to the ARM.

8. The ARM verifies the response from the user. If it is valid, it then retrieves $Cipher^{Conditions-SP1}_{IBEPRE-K^u_{priv-VE}}$, and re-encrypts it under *Conditions-SP2* to generate $Cipher^{Conditions-SP2}_{IBEPRE-K^u_{priv-VE}}$ (the ARM can do the re-encryption as it knows $Cipher^{Conditions-SP1}_{IBEPRE-K^u_{priv-VE}}$, `msk`, *Conditions-SP1*, and *Conditions-SP2*)

9. The ARM stores *Conditions-SP2* + $Cipher^{Conditions-SP2}_{IBEPRE-K^u_{priv-VE}}$, and sends a signed $Cipher^{Conditions-SP2}_{IBEPRE-K^u_{priv-VE}}$ to the IdP

10. The IdP verifies the ARM's signature over $Cipher^{Conditions-SP2}_{IBEPRE-K^u_{priv-VE}}$. If valid, it then sends a signed response message back to $SP2$. Included in the response are $pseudo_a$, $Cipher^{Conditions-SP2}_{IBEPRE-K^u_{priv-VE}}$ + *Conditions-SP2*, $Cipher^{K^u_{pub-VE}}_{VE-m_a}...Cipher^{K^u_{pub-VE}}_{VE-m_c}$, the identity of the ARM, and other messages specified by the FSSO protocol used.

11. $SP2$ verifies the response returned from the IdP. If valid, then $SP2$ knows that $m_a...m_c$ can be recovered with the help of the ARM when *Conditions-SP2* are satisfied

Revocation When an SP, say $SP1$, believes that *Conditions-SP1* are satisfied, it will start the revocation stage:

1. SP sends a signed *Conditions-SP1* to the ARM

2. The ARM verifies the message from SP and checks if it has the same *Conditions-SP1'* stored. If so, it checks if the $SP1$ it is talking to is the same as the identity of $SP1'$ as stated in $Conditions - SP1'$.

[2]The IdP can detect such an open authenticated session with $pseudo_a$ because in the existing FSSO protocols, the request message that $SP2$ generated earlier is actually sent through a redirection from the user to the IdP, thus some authenticated session information (such as cookies) can be passed along to the IdP from the user machine

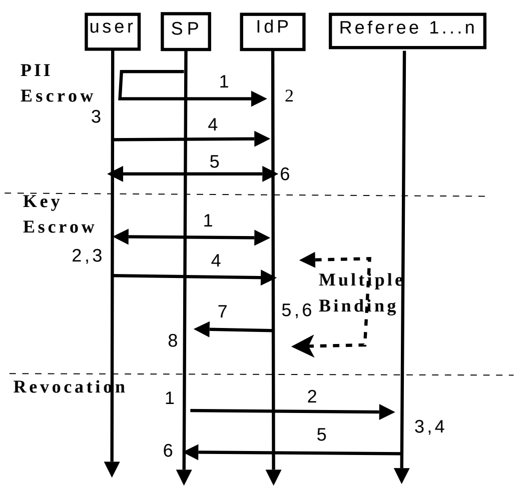

Figure 2: FSSO-PIEMC without trusted ARM

3. If valid, the ARM then verifies if the conditions included in $Conditions - SP1'$ are satisfied.

4. If the policies are satisfied, then the ARM extracts $sk_{Conditions-SP1}$ which is the private key for *Conditions-SP1*.

5. ARM sends $sk_{Conditions-SP1}$ to $SP1$, who can then decrypt $Cipher^{Conditions-SP1}_{IBEPRE-K^u_{priv-VE}}$ to recover $K^u_{priv-VE}$, and then use $K^u_{priv-VE}$ to recover $Cipher^{K^u_{pub-VE}}_{VE-m_a}...Cipher^{K^u_{pub-VE}}_{VE-m_c}$.

5 FSSO-PIEMC without trusted ARM

The above protocol puts too much trust on the ARM - who is also the PKG - to behave honestly. If the ARM is malicious, then users are not protected (as the ARM can easily decrypt all of the re-encrypted $K^u_{priv-VE}$ and obtain the corresponding PII). In this section, the second variant of our FSSO-PIEMC is detailed whereby no trusted ARM is used.

Overview The setup and PII escrow stage are the same. During the multiple binding stage, the user's *extended TPM* platform will perform an execution of `Module2`: given K^u_{pub-VE}, a set of conditions, and other public parameters for the UCH encryption scheme, perform an encryption of $K^u_{priv-VE}$ for n referees under the given conditions. The encrypted $K^u_{priv-VE}$ is then sent to the IdP, who will in turn pass it on to the SP, along with $pseudo_a$, and other necessary information. During revocation, the SP sends the encrypted $K^u_{priv-VE}$ to the n referees who will then verify if all the necessary conditions are satisfied, and if so, help the SP to recover $K^u_{priv-VE}$. With $K^u_{priv-VE}$ recovered, the SP can recover the escrowed PII.

Setup and PII Escrow No changes

Key Escrow Assume a similar scenario to the one used in section 4: an $SP1$ needs to bind a set of escrowed PII to *Conditions-SP1*. For a group of referees R consisting of n referees, each of the referees public encryption keys $K^{ref_i}_{pub-UCH}$ are publicly known. See Figure 2.

1. u_a and IdP engage in a DAA protocol to verify that the user is using a valid TPM platform and to establish a one-time *AIK* key pair ($K^{TPM-u_a}_{verify-AIK}$, $K^{TPM-u_a}_{sign-AIK}$), as well as one-time user signing key $K^{u_a}_{verify}$, $K^{u_a}_{sign}$ to be used for that session.

2. u_a spontaneously forms a designated group T consisting of t referees from a group of all referees R ($T \subset R$)

3. u_a runs `Module2` on her *extended TPM* platform - see Table 2 for details. This module will generate an output $\psi = Cipher^{K^{refs_i}_{pub-UCH}, Conditions-SP1}_{UCH(t,t,n)-K^u_{priv-VE}} + random^{r_i}$ (for $s \in T$, and $r \in [1,n] \setminus T$, $i = 1...n$)

4. u_a's platform generates the proof of correct execution of `Module2`, signs the proof using $K^{TPM-u_a}_{sign-AIK}$, and sends ψ, and the proof to the IdP.

5. After a successful verification of the proof, the IdP generates a one-time pseudonym $pseudo_a$ to identify the user for that particular FSSO session only.

6. The IdP also stores $pseudo_a$, and links it to ψ, K^u_{pub-VE}, and $Cipher^{K^u_{pub-VE}}_{VE-m_a} ... Cipher^{K^u_{pub-VE}}_{VE-m_c}$.

7. The IdP then sends a response message back to $SP1$. Included in the response are $pseudo_a$, ψ, *Conditions-SP1*, $Cipher^{K^u_{pub-VE}}_{VE-m_a} ... Cipher^{K^u_{pub-VE}}_{VE-m_c}$, and other messages specified by the FSSO protocol used.

8. $SP1$ verifies the response returned from the IdP. If valid, then $SP1$ knows that $m_a ... m_c$ can be recovered, with the help of the referees, when *Conditions-SP1* are satisfied.

Input	t (size of T), n (size of R), K^u_{pub-VE}, $K^i_{pub-UCH}$ ($i = 1...n$), *Conditions*
Process	P2.1. Retrieve $K^u_{priv-VE}$ P2.2. Verify that $K^u_{priv-VE}$ is the private key for the input K^u_{pub-VE} P2.3. For $s \in T$, generates $Cipher^{K^{refs_i}_{pub-UCH}, Conditions}_{UCH(t,t,n)-K^u_{priv-VE}}$ P2.4. For $r \in [1,n] \setminus T$, generates $random^{r_i}$
Output	$Cipher^{K^{refs_i}_{pub-UCH}, Conditions}_{UCH(t,t,n)-K^u_{priv-VE}} + random^{r_i}$

Table 2: `Module2` - UCH encryption of $K^u_{priv-VE}$

Multiple Binding When the user goes to another $SP2$ who needs to bind the user's escrowed PII to another set of conditions *Conditions-SP2*, similar operations as those in the key escrow stage are performed, with the exception of step 1 which does not have to be performed again as the generated *AIK* key pair and the user signing key pair are valid throughout the session. In step 5 and 6, the IdP this time will use the generated $pseudo_a$ from the key escrow stage, instead of creating a new one.

Revocation When an SP (say $SP2$) believes that *Conditions-SP2'* are satisfied, it will start the revocation stage:

1. $SP2$ retrieves ψ associated with *Conditions-SP2'*

2. For $i = 1...n$, $s \in T$, and $r \in [1,n] \setminus T$, $SP2$ sends *Conditions-SP2'* + ($Cipher^{K^{refs_i}_{pub-UCH}, Conditions-SP2}_{UCH(t,t,n)-K^u_{priv-VE}}$ or $random^{r_i}$) to the respective n referees (we assume the use of authenticated and confidentiality-protected channel between $SP2$ and the referees)

3. Each referee checks if the identity of the $SP2'$ it is talking to is the same as the one included in the given *Conditions-SP2'*. If this is not the case, stops.

4. Each referees verifies if the conditions included in *Conditions-SP2'* are satisfied.

5. If the policies are satisfied, then each referee decrypts the given ciphertext pieces:

   (a) As part of the decryption algorithm in (Liu et al. 2005), each ref_i ($i \in T$) will be able to verify that it has a well-formed ciphertext piece. Thus, the decryption process will result in a share $P^i_{K^u_{priv-VE}}$. Sends $P^i_{K^u_{priv-VE}}$ to $SP2$.

   (b) For each $referee_i$ ($i \in [1,n] \setminus T$), it will detect that it has received random value (not well-formed ciphertext), thus sends a `reject` message to $SP2$

6. When $SP2$ has received t decrypted shares ($P^{1...t}_{K^u_{priv-VE}}$), it can then recover $K^u_{priv-VE}$. From there on, it can use $K^u_{priv-VE}$ to recover $Cipher^{K^u_{pub-VE}}_{VE-m_a} ... Cipher^{K^u_{pub-VE}}_{VE-m_c}$.

6 Discussion

In this section, an informal security analysis is provided to show how the detailed FSSO-PIEMC protocols (with and without trusted *ARM*) have achieved the security requirements detailed in section 2.1. How well these protocols have achieved the *minimum online computations* property is discussed in the next section.

6.1 Analysis of FSSO-PIEMC with trusted *ARM*

Given the threat environment detailed in section 2.2, the FSSO-PIEMC protocol with a trusted *ARM* achieves the *multiple conditions, zero-knowledge, indirect enforcement of conditions, authenticated PII escrow, conditions-abuse resistant, and forward secrecy* properties as long as (1) the *extended TPM* platform behaves and processses `Module1` as it is expected to, and (2) the underlying cryptographic primitives used are correct and secure, and (3) the *ARM* behaves honestly.

In accordance to the threat model specified in section 2.2, we further specify the actions that malicious users, IdPs, and SPs may do:

- A malicious user u'_a may provide false PII (for escrow)
- As the PII escrow is ultimately bound to a set of conditions, the easiest way to compromise the system is to attempt to bind the escrowed PII

with a set of easy-to-fulfill conditions (for the SPs and IdPs) or hard-to-fulfill conditions (for users). Therefore, we consider the following actions:

  - u'_a may use a hard-to-fulfill *Conditions-x'* during execution of `Module1`, which is not the same with *Conditions-x* originally agreed with an SP_x
  - An SP_x may provide an easy-to-fulfill *Conditions-x'* to an IdP which is not the same with *Conditions-x* originally agreed with a user u_a
  - Similarly, an IdP may give an easy-to-fulfill *Conditions-x'* to the *ARM* which is not the same with the one originally agreed between a user and an SP_x.

- A malicious IdP or SP may attempt to start the revocation stage by asking the *ARM* to extract a decryption key based on a correct but not-satisfied *Conditions-x*, or a false but easy-to-fulfill *Conditions-x'*.

Recall that the *ARM* is trusted in this variant of the protocol.

Multiple conditions The *Multiple conditions* property is achieved due to the re-encryption property of an IBEPRE encryption scheme, the CRPI capability of the anonymous credential system used (Bangerter et al. 2004), as well as the *provable isolated execution* of `Module1`.

During the PII escrow stage, a user's PII, say $m_a...m_c$, are encrypted using a user-generated one-time VE key. The corresponding private key $K^u_{priv-VE}$ is later encrypted using an IBEPRE encryption scheme by executing `Module1` on the user's *extended TPM* platform. The *provable isolated execution* of an *extended TPM* platform allows one to verify that a given output is the result of a correct isolated execution of a known integrity-verified module, based on a given set of verified inputs.

Therefore, given the one-time VE public key K^u_{pub-VE}, the conditions *Conditions-X*, and the public parameters for IBEPRE, the output of `Module1`, $Cipher^{Conditions-x}_{IBEPRE-K^u_{priv-VE}}$ must be an IBEPRE encryption of the corresponding private key $K^u_{priv-VE}$ under *Conditions-X*: this is because at step P1.2 of `Module1` (see Table 1), the retrieved private key $K^{u'}_{priv-VE}$ is checked against the input public key K^u_{pub-VE} to verify that it is the correct private key to encrypt. If for some reasons the retrieved $K^{u'}_{priv-VE}$ has been altered (such as due to storing it in an insecure storage area), the step P1.2 verification will fail.

A malicious user u'_a may also use invalid IBEPRE public parameters and conditions to `Module1`. However, such actions can be detected from the signed proof that the TPM returns - which includes the input and output parameters - see (McCune et al. 2008) for details. Modifications of the codes of `Module1` will also be detected in the similar manner.

Therefore, if $Cipher^{Conditions-x}_{IBEPRE-K^u_{priv-VE}}$ correctly encrypts $K^u_{priv-VE}$, then the re-encrypted ciphertext $Cipher^{Conditions-x_2}_{IBEPRE-K^u_{priv-VE}}$ generated by an *ARM* must also hide the same key $K^u_{priv-VE}$, but this time under *Conditions-X*$_2$, due to the re-encryption property of an IBEPRE scheme and due to the assumption that the *ARM* is honest. As the user's PII can only be decrypted using $K^u_{priv-VE}$, and as $K^u_{priv-VE}$ can be re-encrypted multiple times, each bound to a different set of conditions, we have therefore indirectly bound the encrypted PII to multiple sets of conditions. In addition, the FSSO-PIEMC protocol is designed such that users and SPs can either have a set of pre-agreed conditions, or have the conditions determined on-line as an FSSO session starts.

Due to these reasons, we therefore argue that the detailed FSSO-PIEMC in section 4 achieves the *Multiple Conditions* property.

Zero-knowledge This property is achieved due to the confidentiality protection property of the VE and IBEPRE encryption schemes. Given that the VE encryption is secure - see (Camenisch & Shoup 2003*a*) for security proofs - the IdPs and SPs, who possess $Cipher^{K^u_{pub-VE}}_{VE-m_a}$ ... $Cipher^{K^u_{pub-VE}}_{VE-m_c}$ will not be able to recover $m_a...m_c$ as they do not have the associated decryption key $K^u_{priv-VE}$.

Similarly, given that the IBEPRE encryption used - such as (Green & Ateniese 2007, Chu & Tzeng 2007, Matsuo 2007) - is secure, the IdPs and SPs who possess $Cipher^{Conditions-X}_{IBEPRE-K^u_{priv-VE}}$ (and its re-encrypted values) will not be able to recover $K^u_{priv-VE}$ as they do not have the necessary decryption keys.

The value of $K^u_{priv-VE}$ can only be recovered when the *ARM* provides the corresponding decryption key to an *authorised* entity. As the *ARM* is honest, a decryption key, say $sk_{Conditions\text{-}X}$ can only be recovered when the *ARM* is satisfied that the associated set of conditions *Conditions-X* is fulfilled, and that the entity requesting the decryption is the same as the one stated in *Conditions-X*. In other words, $K^u_{priv-VE}$ (and subsequently $m_a...m_c$) can only be recovered by an *authorized SP*. This property fits our definition of the *zero-knowledge* property.

Indirect Enforcement of Conditions This property is achieved due to the property of any identity-based encryption scheme: given an id_1 as the public encryption key, the `Extract` operation will output the decryption key sk_{id_1} that is valid for id_1 only. In our FSSO-PIEMC protocol, the enforcement of conditions is exercised during the revocation stage. If a malicious SP_x gives an unfulfilled, but correct, set of conditions, say *Conditions-x* to the *ARM*, an honest *ARM* will verify that *Conditions-x* are not satisfied, thus refuse to provide the decryption key.

If the SP_x provides a bogus set of easy-to-fulfill conditions, say *Conditions-x'*, the *ARM* may verify that the conditions are satisfied, and thus release the decryption key $sk_{conditions-x'}$. However, this decryption key is useless because $K^u_{priv-VE}$ was encrypted under *Conditions-x* which can only be decrypted using $sk_{conditions-x} \neq sk_{conditions-x'}$.

Therefore, given that *ARM* is an honest entity, the proposed FSSO-PIEMC protocol achieves the *indirect enforcement of conditions* property.

Authenticated PII escrow This property is achieved due to the CRPI property from the anonymous credential system (Bangerter et al. 2004), as well as the *multiple conditions* property that our proposed FSSO-PIEMC protocol provides. The CRPI property (see Introduction) allows users' PII to be encrypted in such way that the recipient can verify if the given ciphertext correctly encrypts some user's PII that have been certified by a trusted certificate issuer. A successful execution of PII-escrow stage means that the IdP is convinced that the received $Cipher_{VE-m_a}^{K_{pub-VE}^u}$... $Cipher_{VE-m_c}^{K_{pub-VE}^u}$ indeed hide $m_a...m_c$.

Combined with the *multiple conditions* property of FSSO-PIEMC (explained earlier), when one of the associated set of conditions is satisfied, an SP will be able to recover a correct $K_{priv-VE}^u$, which can thus be used to recover $m_a...m_c$ successfully.

Conditions-Abuse resistant The proposed FSSO-PIEMC with trusted *ARM* achieves the *conditions-abuse resistance* property due to the design of the key-escrow and the multiple-binding stage, as well as the *indirect enforcement of conditions* property explained earlier. In step 7 of the key-escrow stage, the *ARM* verifies the generated TPM proof of correct execution of `Module1`. If u'_a used wrong *Conditions-x'* as input to `Module1`, then the validation of the proof will fail because the *ARM* will use, as one of its inputs, the value of *Conditions-x* provided by the IdP in validating the TPM proof.

Similarly, if the *ARM* receives a wrong set of conditions *Conditions-x'* (either due to the SP giving wrong conditions *Conditions-x'* to the IdP, or directly from the IdP itself), the validation of the TPM proof will also fail as the *ARM* will use the the incorrect *Conditions-x'* value as one of the inputs in verifying the given TPM proof. The TPM-proof validation will only succeed if the given condition string from the IdP is exactly the same as the one the user used as input to `Module1`.

During the multiple binding stage, steps 4 to 8 ensure that the VE private key is correctly re-encrypted to the correct conditions. Assume we need to re-encrypt an IBEPRE encrypted $K_{priv-VE}^u$ from *Conditions-x* to *Conditions-y*. In step 4, if the IdP has given an invalid *Conditions-x'*, the *ARM* will be able to detect it and stops the protocol execution. In steps 5-7, if the IdP has given an invalid conditions *Conditions-y'*, the user u_a will be able to detect it too (signature verification fails). Therefore, a re-encryption of $K_{priv-VE}^u$ will not be performed unless the SP and IdP provides valid *Conditions-x* and *Conditions-y* to the *ARM*.

Finally, as shown earlier, the *indirect enforcement of conditions* property prevents malicious SPs or IdPs from providing either correct but not fulfilled, or incorrect but easy-to-fulfill conditions to the *ARM* for PII revocation purpose.

Due to these reasons, and since the *ARM* is honest, we therefore argue that the proposed FSSO-PIEMC protocol also achieves the *Conditions-abuse resistant* property.

Forward secrecy This property is achieved due to the one-time use property of the VE key in FSSO-PIEMC, and also due to the non-deterministic nature of the VE encryption scheme (Camenisch & Shoup 2003*b*). Informally, given the same message to encrypt, a non-deterministic encryption scheme produces a different encryption result for each round of encryption.

Assume an SP_x who has successfully revealed PII m_x of a user u_a. This SP_x also has a collection of l encrypted PII of many other users: $C = Cipher_{VE-m_a,...m_z}^{K_{pub-VE}^{u_{1...l}}}$ Assume that in this collection, a subset of it is made up of h encryptions of the same PII m_x of user u_a encrypted under *different* one-time VE keys: $E = Cipher_{VE-m_x}^{K_{pub-VE}^{u_{1...h}}}$ $(E \subset C)$. The *forward secrecy* property is violated if an SP can pick out the subset E from C while knowing only m_x. However, as each encryption of m_x uses a different VE key, and due to the non-deterministic nature of the VE scheme, the resulting ciphertext $Cipher_{VE-m_x}^{K_{pub-VE}^{u_1}}$ should be indistinguishable from another ciphertext $Cipher_{VE-m_x}^{K_{pub-VE}^{u_2}}$ of the same m_x but encrypted using a different key $K_{pub-VE}^{u_2}$, or, from another ciphertext $Cipher_{VE-m_y}^{K_{pub-VE}^{y_1}}$, which is an encryption of a different data item m_y of different user. Of course, this also means that users must generate the one-time VE keys randomly.

Therefore, assuming that users choose their one-time VE keys in a random manner, the *forward secrecy* property as defined in section 2.1 is achieved.

6.2 Analysis of FSSO-PIEMC without trusted *ARM*

Given the threat environment detailed in section 2.2, the FSSO-PIEMC protocol without a trusted *ARM* achieves the *multiple conditions, zero-knowledge, indirect enforcement of conditions, authenticated PII escrow, conditions-abuse resistance, and forward secrecy* properties as detailed in section 2.1 as long as (1) the *extended TPM* platform behaves as it is expected to, (2) the underlying cryptographic primitives used are correct and secure, and (3) there is at least one honest referee in the designated group of referees T.

In addition to the actions that malicious users, IdPs, and SPs can do detailed in section 6.1, we also add the following allowable actions of malicious referees:

- Malicious referees in R_{dh} may state that a set of conditions is satisfied while it actually is not.
- Malicious referees may provide a set of invalid yet easy-to-fulfill conditions to all referees in R in the hope of gathering enough t decrypted shares to recover $K_{priv-VE}^u$.

Multiple conditions The *Multiple conditions* property is achieved due to the *provable isolated execution* of `Module1`. During the PII escrow stage, a user's PII, say $m_a...m_c$, are encrypted using a one-time VE key, whose private key portion is later encrypted using the UCH encryption scheme by the execution of `Module2` on the user's *extended TPM* platform. Similar arguments as provided in section 6.1 applies: step P2.2 (see Table 2) ensures that the output of `Module2` correctly encrypts the correct VE private key in relation to the input public key K_{pub-VE}^u. Invalid input (such as providing invalid *Conditions'*, or invalid public keys of the referees) will also be detected through the verification of the provided TPM proof.

The multiple binding stage is simply a new encryption of $K_{priv-VE}^u$ under new conditions. Therefore, it again relies on the *provable isolated execution* of the user's *extended TPM* platform to behave correctly. The FSSO-PIEMC without trusted *ARM* also allows users and SPs to have either pre-agreed conditions, or have the conditions determined on-line as an FSSO session starts.

Zero-knowledge This property is achieved due to the confidentiality protection provided by the VE and UCH encryption schemes. Given that the VE encryption (Camenisch & Shoup 2003*a*) is secure, the IdPs and SPs, who possess $Cipher_{VE-m_a}^{K_{pub-VE}^u}$...

$Cipher_{VE-m_c}^{K^u_{pub-VE}}$ will not be able to recover $m_a ... m_c$ as they do not have the associated decryption key $K^u_{priv-VE}$. Similarly, given that the UCH encryption used (Liu et al. 2005) is secure, the IdPs and SPs who possess ψ will not be able to recover $K^u_{priv-VE}$.

The value of $K^u_{priv-VE}$ can only be recovered when all t referees in T agree that the associated set of conditions *Conditions-x* is fulfilled (since we use the $UCH(t,t,n)$ encryption scheme), and that the entity requesting the decryption is the same as the one stated in *Conditions-x*. As we assume the existence of at least 1 honest referee in T, it is therefore obvious that $m_a ... m_c$ can only be recovered by an *authorized SP-x*: with $UCH(t,t,n)$, we can handle up to $t-1$ dishonest referees. Thus even if referees collude, they will only get a maximum of only $t-1$ decrypted shares, which are not enough to recover $K^u_{priv-VE}$.

Indirect Enforcement of Conditions This property is achieved due to the *encryption under a label* property of the UCH scheme (Liu et al. 2005) whereby a message is encrypted under a label, in our case, under a the *Conditions-x* string. For a successful decryption, the algorithm requires the use of the *same* label *Conditions-x* as used during the encryption, otherwise, the decryption algorithm will result in an error.

If an SP_x provides a bogus set of easy-to-fulfill conditions, say *Conditions-x'*, the referees may agree that the conditions are satisfied. However, the decryption process will output an error. Similarly, if SP_x provides correct but not satisfied *Conditions-x* to the refereees, those referees in R_{dh} may 'pretend' that they are satisfied, and thus provide the decrypted shares. However, again, an honest referee will not continue the decryption process, resulting in not enough shares to recover $K^u_{priv-VE}$.

As we assume the existence of at least one honest referee in T, the *indirect enforcement of conditions* property is therefore achieved.

Authenticated PII escrow This property is achieved through the same reasoning as the one used in section 6.1.

Conditions-Abuse resistant The proposed FSSO-PIEMC without trusted *ARM* achieves the *conditions-abused resistant* property due to the *provable isolated execution* property of user's *extended TPM* platform and the *indirect enforcement of conditions* property explained earlier. In step 3-4 of the key escrow (and multiple binding) stage, the IdP will try to verify the return TPM proof using the given condition string *Conditions-x* from the SP. Therefore, it will be able to detect if the user has used an incorrect *Conditions-x'* as an input to `Module2`. Similarly, if an SP sends an incorrect set of conditions *Conditions-x'* to the IdP, the verification of the TPM proof will also fail. Therefore, the proposed FSSO-PIEMC protocol also achieves the *conditions-abuse resistant* property.

Forward secrecy Similar reasoning as explained in section 6.1 applies.

7 Performance Analysis of FSSO-PIEMC

We measure the performance of FSSO-PIEMC using the number of cryptographic operations that have to be performed by users, IdPs, SPs, *ARM*, and the referees. We show that our protocol provides a significantly better efficiency as compared to the existing approach (Bangerter et al. 2004).

We base the calculation on the number of signature creation (*Sign*), signature verification (*Verify*), commitments generations, *PK* operations for proving correct commitments (*PK-comm*), *PK* operations for proving correct VE encryptions (*PK-VE*), execution of the DAA protocol, the encryptions (*Enc*), and decryptions (*Dec*) operations of the VE and UCH schemes, and the IBEPRE operations (encryption, decryption, private key extraction, and re-encryption). Table 3 and Table 4 shows the total cryptographic operations that have to be performed.

Of these operations, the *PK-VE*, *PK-comm*, *Enc*(UCH), and DAA, consume the most computational resources as they require numerous computationally-intensive modular exponentiations (modex). As an example, based on a rough estimate, the *PK-VE* operation requires a prover (e.g. a user) to perform roughly 10 modex, while a verifier (e.g. an IdP) needs to perform approximately 13 modex - see (Camenisch & Shoup 2003*b*) for details. A UCH encryption takes approximately $5t$ modex (t refers to the number of designated referees in T).

7.1 Improved Performance with FSSO-PIEMC

In both variants of the FSSO-PIEMC protocol (with and without the trusted *ARM*), the first round interaction with an SP triggers the PII escrow and key escrow stage. If we use optimization technique whereby the user and IdP perform the PII escrow stage offline, then we can deduct 'd commitments + d VE encryptions + d *PK-comm* + d *PK-VE*' operations from the users side, and 'd *PK-Comm* + d *PK-VE*' operations from the IdP side. For each of the next r-round interactions with the other SPs in the session, we only need to perform the multiple-binding stage. The revocation stage is only invoked as necessary.

FSSO-PIEIMC with trusted *ARM* In the worst case scenario, the proposed FSSO-PIEMC suffers from inefficient first-round operation due to the required *PK-comm, PK-VE*, and DAA operations. However, the efficiency of the subsequent rounds is massively improved, especially for the users who only need to do one signature generation and one signature verification - a very useful property for users with low-powered devices. Of course, the majority of operations is now transfered to the *ARM* who has to perform a re-encryption (which may include a private key extraction and a re-encryption key generation - see Appendix A.2) for each of the r SPs. However, even so, they are all based on efficient elliptic curve cryptography operations. Besides, it is very likely that an *ARM* would operate using a system with a considerable amount of computational powers. In the optimum case, the first-round only requires one intensive cryptographic operations: DAA to be performed between a user and an *ARM*.

FSSO-PIEIMC without trusted *ARM* The worst case scenario performance is roughly equivalent to the FSSO-PIEMC with trusted *ARM*. However, for each of the subsequent rounds, while there is no bottleneck at the *ARM* (no *ARM* is involved), the user has to perform one *Enc*(UCH) encryption operation. Providing a protocol that works in a stronger threat environment (assuming no trusted *ARM*) comes at the cost of a decreased performance at the users' side.

Players		User	IdP	SP	*ARM*
PII + Key Escrow (1st SP)	Max	d(commit + VE + *PK-comm* + *PK-VE*) + 1 DAA + 1 *Enc*($IBEPRE$) + 1 *Sign*	d(*PK-comm* + *PK-VE*) + 2 *Sign* + 2 *Verify*	1 *Sign* + 1 *Verify*	1 DAA + 1 *Sign* + 1 *Verify*
	Opt	1 DAA + 1 *Enc*($IBEPRE$) + 1 *Sign*	2 *Sign* + 2 *Verify*	1 *Sign* + 1 *Verify*	1 DAA + 1 *Sign* 1 + *Verify*
Each of the next r SPs		1 *Sign* + 1 *Verify*	2 *Sign* + 2 *Verify*	1 *Sign* + 1 *Verify*	2 *Sign* + 2 *Verify* + 1 `Re-encrypt`
Revocation				1 *Sign* + 1 *Dec*($IBEPRE$) + d *Dec*(VE)	1 *Verify* + 1 `Extract`

Table 3: Performance Summary for FSSO-PIEMC with trusted *ARM*

Players		User	IdP	SP	Referee
PII + Key Escrow (1st SP)	Max	d(commit + VE + *PK-comm* + *PK-VE*) + 1 DAA + 1 *Enc*(UCH) + 1 *Sign*	d(*PK-comm* + *PK-VE*) + 1 DAA + 1 *Sign* + 2 *Verify*	1 *Sign* + 1 *Verify*	
	Opt	1 DAA + 1 *Enc*(UCH) + 1 *Sign*	1 DAA + 1 *Sign* + 2 *Verify*	1 *Sign* + 1 *Verify*	
Each of the next r SPs		1 *Enc*(UCH) + 1 *Sign*	1 *Sign* + 1 *Verify*	1 *Sign* + 1 *Verify*	
Revocation				d *Dec*(VE)	1 *Dec*(UCH)

Table 4: Performance Summary for FSSO-PIEMC without trusted *ARM*

Players	User	IdP	SP	*ARM*
FSSO-PIEMC with trusted *ARM*	d(*PK-comm* + *PK-VE* + VE + commit) + 1 DAA	d(*PK-comm* + *PK-VE*)		1 DAA + r `Re-encrypt`
Existing Approach	$r \times d$(*PK-comm* + *PK-VE*)		$r \times d$(*PK-comm* + *PK-VE*)	
FSSO-PIEMC without *ARM*	d(*PK-comm* + *PK-VE* + VE + commit) + 1 DAA + r(*Enc*(UCH))	d(*PK-comm* + *PK-VE*) + 1 DAA		

Table 5: Performance Comparison between FSSO-PIEMC and existing approach (Bangerter et al. 2004) for interactions with r-number of SPs in a session

Comparison to Existing appraoch Assume we need to escrow d-number of PII. In the existing approach, every single interaction with a different SP requires the execution of the PII escrow operation: for d PII to escrow, a user has to generate d commitments and d VE encryptions. In addition, the user and the SP have to perform d(*PK-comm* + *PK-VE*). So, if a user in a session interacts with r number of SPs, a user has to perform rd(commitments + VE + *PK-comm* + *PK-VE*) operations, while the each SP has to perform rd(*PK-comm*+*PK-VE*).

To simplify the comparison, let us just consider the main cryptographic operations: VE, commitments generations, *PK-comm*, *PK-VE*, *Enc*(UCH), DAA, and IBEPRE re-encryption. See Table 5 for a summary of the performance comparison. In comparison with the existing approach, our FSSO-PIEMC with trusted *ARM* improves the performance by roughly a factor of r: *regardless of the number of SPs a user interacts in a session, both variants of the FSSO-PIEMC protocol only have to perform the resources-intensive cryptographic operations (*PK-comm, PK-VE, DAA*)* ***once*** *only*. While the *ARM* still has to perform roughly r IBEPRE re-encryption, it requires a much less computational resources as compared to performing rd(commitments + VE + *PK-comm* + *PK-VE*).

The performance comparison with the FSSO-PIEMC without trusted *ARM* appears to be roughly comparable: while users only have to perform d(*PK-VE* + *PK-comm* + VE + commit), they have to perform r(*Enc*(UCH)), which is another resources-intensive operations. However, our protocol is still an improvement as we *do not need* any trusted *ARM* anymore.

Such an improved performance is obtained from using ACS in an FSSO setting, *and* removing the need to perform *PK* by replacing it with the use of an *extended TPM* platform whereby the *PK* operations can be reduced to only verifying the TPM-signed PCR registry value, input, output, and other necessary parameters.

8 Conclusion and Future Work

Two variants of FSSO-PIEMC protocols are proposed, with and without trusted *ARM*, including their security properties and their performance improvement over the existing approach. Future work involve investigating into the possibility of adding the use of the *extended TPM* technology at the SPs, IdPs, and/or referees' sides to possibly improve efficiency, especially during the revocation stage. While at this stage, in the context of addressing the *multiple conditions* and *trusted ARM* problems, we do not see any substantial performance gains from doing so, further research is still required. Another future work involves relaxing the threat model to allow collusion between users and referees. In the FSSO-PIEMC without trusted *ARM*, a referee in T who colludes with users can always reject conditions fulfillment, thus allowing irrevocable users' PII .

References

Baek, J., Newmarch, J., Safavi-Naini, R. & Susilo, W. (2004), A survey of identity-based cryptography, *in* 'AUUG 2004'.

Bangerter, E., Camenisch, J. & Lysyanskaya, A. (2004), A cryptographic framework for the controlled release of certified data, *in* B. Christianson, B. Crispo, J. A. Malcolm & M. Roe, eds, 'Security Protocols Workshop', Vol. 3957 of *LNCS*, Springer, pp. 20–42.

Bhargav-Spantzel, A., Camenisch, J., Gross, T. & Sommer, D. (2006), User centricity: a taxonomy and open issues, *in* A. Juels, M. Winslett & A. Goto, eds, 'DIM', ACM, pp. 1–10.

Brickell, E. F., Camenisch, J. & Chen, L. (2004), Direct anonymous attestation, *in* V. Atluri, B. Pfitzmann & P. D. McDaniel, eds, 'ACM Conference on Computer and Communications Security', ACM, pp. 132–145.

Camenisch, J. & Lysyanskaya, A. (2002), A signature scheme with efficient protocols, *in* S. Cimato, C. Galdi & G. Persiano, eds, 'SCN', Vol. 2576 of *LNCS*, Springer, pp. 268–289.

Camenisch, J. & Lysyanskaya, A. (2004), Signature schemes and anonymous credentials from bilinear maps, *in* M. K. Franklin, ed., 'CRYPTO', Vol. 3152 of *LNCS*, Springer, pp. 56–72.

Camenisch, J. & Shoup, V. (2003*a*), Practical verifiable encryption and decryption of discrete logarithms, *in* D. Boneh, ed., 'CRYPTO', Vol. 2729 of *LNCS*, Springer, pp. 126–144.

Camenisch, J. & Shoup, V. (2003*b*), Practical verifiable encryption and decryption of discrete logarithms (slides), *in* 'The 7th Workshop on Elliptic Curve Cryptography (ECC 2003)', University of Waterloo, Canada.

Chen, L. (2005), *Direct Anonymous Attestation*, Hewlett Packard Laboratory. `https://www.trustedcomputinggroup.org/news/presentations/051012_DAA-slides.pdf`.

Chu, C.-K. & Tzeng, W.-G. (2007), Identity-based proxy re-encryption without random oracles, *in* J. A. Garay, A. K. Lenstra, M. Mambo & R. Peralta, eds, 'ISC', Vol. 4779 of *LNCS*, Springer, pp. 189–202.

Davida, G. I., Frankel, Y., Tsiounis, Y. & Yung, M. (1997), Anonymity control in e-cash systems, *in* R. Hirschfeld, ed., 'Financial Cryptography', Vol. 1318 of *LNCS*, Springer, pp. 1–16.

Dell (2004), *Securing Network-based Client Computing: User and Machine Security*, Dell. `http://www.dell.com/downloads/global/vectors/2004_tpm.pdf`.

George, P. (2004), User authentication with smart cards in trusted computing architecture, *in* H. R. Arabnia, S. Aissi & Y. Mun, eds, 'Security and Management', CSREA Press, pp. 25–31.

Green, M. & Ateniese, G. (2007), Identity-based proxy re-encryption, *in* J. Katz & M. Yung, eds, 'ACNS', Vol. 4521 of *Lecture Notes in Computer Science*, Springer, pp. 288–306.

Intel (2007), *Intel Trusted Execution Technology*, Intel.

Liu, J. K., Tsang, P. P., Wong, D. S. & Zhu, R. W. (2005), Universal custodian-hiding verifiable encryption for discrete logarithms, *in* D. Won & S. Kim, eds, 'ICISC', Vol. 3935 of *LNCS*, Springer, pp. 389–409.

Lockhart, H., Andersen, S. et al. (2006), *Web Services Federation Language (WS-Federation).*

Matsuo, T. (2007), Proxy re-encryption systems for identity-based encryption, *in* T. Takagi, T. Okamoto, E. Okamoto & T. Okamoto, eds, 'Pairing', Vol. 4575 of *Lecture Notes in Computer Science*, Springer, pp. 247–267.

McCune, J. M., Parno, B., Perrig, A., Reiter, M. K. & Isozaki, H. (2008), Flicker: an execution infrastructure for TCB minimization, *in* J. S. Sventek & S. Hand, eds, 'EuroSys', ACM, pp. 315–328.

Mitchell, C. J. (2005), *Trusted Computing*, Institution of Electrical Engineers, chapter 1.

OASIS (2005*a*), *Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0*, OASIS. OASIS Standard.

OASIS (2005*b*), *Security and Privacy Considerations for the OASIS Security Assertion Markup Language (SAML) V2.0*, OASIS.

Reid, J. F. (2007), Enhancing security in distributed systems with trusted computing hardware, PhD thesis, Queensland Univeresity of Technology.

Shao, J., Xing, D. & Cao, Z. (2008), Identity-based proxy re-encryption schemes with multiuse, unidirection, and cca security, Cryptology ePrint Archive 2008/103, Department of Computer Science and Engineering, Shanghai Jiao Tong University. `http://eprint.iacr.org/2008/103`.

Suriadi, S., Foo, E. & Josang, A. (2008), 'A user-centric federated single sign-on system', *Journal of Network and Computer Applications (In Press)* . `http://dx.doi.org/10.1016/j.jnca.2008.02.016`.

Suriadi, S., Foo, E. & Smith, J. (2008*a*), Conditional privacy using re-encryption. IFIP Network and System Security - NSS 2008 (To appear). `http://eprints.qut.edu.au/archive/00014076/`.

Suriadi, S., Foo, E. & Smith, J. (2008*b*), A user-centric protocol for conditional anonymity revocation. To appear at 5th International Conference on Trust, Privacy & Security in Digital Business - TrustBus 2008. `http://eprints.qut.edu.au/archive/00013123/`.

TCG (2007), *Trusted Platform Module Specification - Part 1 Design Principles Version 1.2 Revision 103*, TCG.

A Background Material

In this section, some background materials are explained so that readers can understand how they are used in the FSSO-PIEMC system. For further details (including the algorithms and security proofs of the cryptographic schemes used), readers should consult the cited references.

A.1 Anonymous Credential System

In an anonymous credential system proposed in (Bangerter et al. 2004), a certificate $Cert$ issued to a user u_a is a *signature* of a certificate issuer $CertIssuer$ over a set of PII $(m_a....m_e)$: $Cert = Sign(m_a, m_b, ...m_e; K_{sign}^{CertIssuer})$. A $Cert$ in an anonymous credential system is *private* to the user. Therefore, the concept of a 'certificate' in an anonymous credential system is different from the 'usual' certificate (such as X509 certificate). This certificate can be verified using the certificate issuer public verification key $K_{verify}^{CertIssuer}$. In this paper, one of the most important capabilities of the anonymous credential system (Bangerter et al. 2004) for a successful FSSO-PIEMC (the CRPI capability) is explained.

Assume we need to escrow a user's email address and tax file number, represented by the PII m_a and m_b, while revealing $m_c...m_e$ to a service provider $SP1$. The PII m_a and m_b are first *blinded* using a commitment scheme: $c_a = Commit(m_a, r)$, $c_b = Commit(m_b, r')$. Then, we encrypt the value of m_a and m_b (which are hidden in c_a and c_b respectively) using the Camenisch-Shoup verifiable encryption scheme (VE) (Camenisch & Shoup 2003*a*) under an ARM public encryption key, along with a set of $Conditions$: $Cipher_{VE-m_a}^{K_{public-VE}^{ARM}} = Enc_{VE}(m_a; Conditions; K_{public-VE}^{ARM})$ (similar operation applied to data item m_b).Here, $Conditions$ is just an input string representing the conditions under which the encrypted PII can be decrypted. Then, a PK is executed to prove that c_a, c_b are the commitments of m_a, m_b contained in $Cert$ issued by a certificate issuer $CertIssuer$. Such PK is achieved by using the *proof of knowledge of a signature on committed messages* technique based on one of the Camenisch-Lysyanskaya signature schemes - see (Camenisch & Lysyanskaya 2002, 2004) for details. This PK also proves that $Cipher_{VE-m_a}^{K_{public-VE}^{ARM}}$ and $Cipher_{VE-m_b}^{K_{public-VE}^{ARM}}$ are encryptions of m_a, m_b hidden in c_a, c_b, under the ARM public key. We can denote such PK as follows:

$$\begin{aligned}
&PK\{(Cert, m_a, m_b) : c_a = Commit(m_a, r) \wedge \\
&c_b = Commit(m_b, r) \wedge \\
&VerifySign(m_a, .., m_e; K_{verify}^{CertIssuer}) = 1 \wedge \\
&Cipher_{VE-m_a} = Enc_{CSVE}(m_a; Conditions; \\
&K_{public-VE}^{ARM}) \wedge \\
&Cipher_{VE-m_b} = Enc_{CSVE}(m_b; Conditions; \\
&K_{public-VE}^{ARM})\}
\end{aligned} \tag{5}$$

At the end of the above PK, $SP1$ will (or will not) be convinced that the escrowed PII hidden in $Cipher_{CSVE-m_a}$ and $Cipher_{CSVE-m_a}$ contain the correct m_a and m_b w.r.t $Cert$, while not learning the plaintext value of m_a or m_b. However, note that $Cipher_{CSVE-m_a}$ and $Cipher_{CSVE-m_b}$ are strictly bound to $Conditions$ - hence the *multiple conditions* problem. Our FSSO-PIEMC addresses this problem.

A.2 Identity-based Proxy Re-encryption Scheme

An identity-based encryption scheme (IBE) allows the use of arbitrary string (called id) as the public key to encrypt a message. A proxy re-encryption scheme (PRE) allows a person (normally called a 'proxy') who possesses a correct re-encryption key to transform a ciphertext $Cipher_{PRE-m}^{A}$ encrypted under an entity A to another ciphertext $Cipher_{PRE-m}^{B}$ encrypted under another entity B's public key without the proxy learning the value of m. We denote such a re-encryption key $rk_{A\rightarrow B}$. Therefore, an identity-based proxy re-encryption scheme (IBEPRE) scheme allows a proxy who has the re-encryption key $rk_{id_1\rightarrow id_2}$ to transform a ciphertext $Cipher_{IBEPRE-m}^{id_1}$ to $Cipher_{IBEPRE-m}^{id_2}$.

As defined in (Green & Ateniese 2007), an IBEPRE scheme consists of the following operations:

- `Setup(`1^λ`, MaxLevels)`: on input of a security parameter 1^λ and `MaxLevels` (the maximum number of consecutive re-encryptions allowed), outputs the public system parameters `params` and master secret key `msk`.
- `KeyGen(params, msk,` id`)`: on input of the system parameters, master secret key, and an identity $id \in \{0,1\}^*$, outputs the secret decryption key sk_{id} for the corresponding id.
- `Encrypt(params,` id, m`)`: on input of a message to encrypt m, `params`, and an id, output $Cipher_{IBEPRE-m}^{id}$.
- `Decrypt(params,` sk_{id}, $Cipher_{IBEPRE-m}^{id}$`)`: on input of a ciphertext $Cipher_{IBEPRE-m}^{id}$, the secret decryption key sk_{id}, and system parameters `params`, output the decrypted message m or $\perp$.
- `RKGen(params,` sk_{id_1}`,` id_1`,` id_2`)`: on input of the system parameters `params`, two identities id_1, id_2, and the secret decryption key sk_{id_1} corresponding to id_1, outputs a re-encryption key $rk_{id_1\rightarrow id_2}$
- `Reencrypt(params,` $rk_{id_1\rightarrow id_2}$`,` $Cipher_{IBEPRE-m}^{id_1}$`)`: on input of a re-encryption key $rk_{id_1\rightarrow id_2}$, a ciphertext $Cipher_{IBEPRE-m}^{id_1}$ encrypted under id_1, and system parameters `params`, outputs a re-encrypted ciphertext $Cipher_{IBEPRE-m}^{id_2}$ encrypted under id_2

Several existing IBEPRE schemes include (Green & Ateniese 2007, Chu & Tzeng 2007, Matsuo 2007, Shao et al. 2008). While they have different algorithms and varying security protections, they have more or less the same set of operations as defined above. The use of IBEPRE in the FSSO-PIEMC system is such that any IBEPRE schemes that provide the above six main operations with chosen ciphertext security can be used.

A.3 Universal Custodian-Hiding Encryption (UCH)

Consider a set of public parameters: for $i = 1, ..., n$, there exists (1) a group N consisting of n members and their corresponding public keys $K_{pub-UCH}^{member_i}$, and (2) spontaneously chosen values of t and k, such that $1 \leq k \leq t \leq n$, and (3) a label (such us a condition string $Conditions$). To encrypt a message m using the UCH scheme (Liu et al. 2005) under the label $Conditions$, a user u_a spontaneously designates t members of N to form a group T ($T \in N$). The UCH scheme (Liu et al. 2005) allows one to encrypt m such that any k out of the t designated members of T have to work together to decrypt it, while hiding the identity of the members of T. If all members of T have to work together ($k = t$), then we have $UCH(t, t, n)$. The identities of the members of T are *hidden*, that is, a recipient of the ciphertext will not be able to learn which of the members of N have been spontaneously chosen by u_a to be the member of T. A user u_a can form the group T consisting of different members with each encryption.

For members of T, t well-formed ciphertext pieces will be generated, each encrypted using the corresponding member's public keys. For members of $N \notin inT$, $n-t$ random values are chosen from specific domains such that they are indistinguishable from the well-formed ones. Regardless, there will be a total of n ciphertext pieces (well-formed + random): $Cipher^{K^s_{pub-UCH-Enc},Conditions}_{UCH(k,t,n)-m} + random^r$ (for $s \in T$, and $r \in [1,n] \backslash T$). This encryption scheme also has its corresponding zero-knowledge proof (PK) system to prove that these n ciphertext pieces correctly encrypt m in relation to some known values γ and δ, such that $\gamma^m = \delta$, and that at least k members of T have to work together to decrypt it without the verifier learning the identity of the members of T. However, as we shall see in section 5, with the use of trusted computing technologies, we do not have to execute the UCH's PK in the proposed FSSO-PIEMC protocol. Further details of this encryption scheme are provided in (Liu et al. 2005). We will also show how UCH can be used in our FSSO-PIEMC protocol to address the *trusted ARM* problem.

A.4 Secure Computing Platform

Trusted computing hardware allows one to reasonably assure that a system will behave as it is expected to, and that it will enforce some security policies (confidentiality, integrity, availability) even in the presence of some physical or logical interference (Mitchell 2005, Reid 2007). An example of such hardware includes trusted platform module (TPM), secure processor, and smartcard.

TPM A TPM (TCG 2007) is a small silicon device that is embedded to a computer device. A TPM securely stores certificates, encryption and signature keys, and passwords. A TPM can perform cryptographic functions, generate random numbers, and, most importantly for our FSSO-PIEMC protocol, allow the attestation of the platform software configuration's *integrity*. When a software module is loaded, its cryptographic hash value (or 'fingerprint') is calculated and added to the Platform Configuration Register (PRC) component inside a TPM. A remote entity can verify if a correct software module is loaded by verifying the value of the PCR register, signed by the TPM.

Each genuine TPM is required by the standard (TCG 2007) to have an embedded (asymmetric) endorsement key EK which never leaves the TPM shielded environment. To prove that a TPM is genuine, the TPM has to prove the knowledge of this EK. For privacy, we only want an entity to learn that it is interacting with a genuine TPM, but not exactly which TPM, otherwise, each interaction can be trivially linked. However, in achieving many of the TPM services, a TPM needs to have an authenticated signature key pair so that messages that it produces can be verified. For example, to verify if a given PCR value is valid, the TPM has to sign the PCR value using its authenticated signing key. If privacy is a concern, then a different signature key pair has to be generated for each session. To convince a party that the per-session generated signature key pair (also known as Attestation Identity Key AIK) is generated from a valid TPM, a technique has been developed to allow a TPM to prove the link between the generated AIK with the embedded EK (which confirms the status of a genuine TPM) without the verifier learning the value of the EK (Chen 2005). Such a technique is called the Direct Anonymous Attestation - DAA (Brickell et al. 2004).

Therefore, by using DAA (Brickell et al. 2004), one can generate many per-session AIKs, thus achieving the unlinkability property amongst TPM transactions. AIK is a signature key, and should never be used for encryption, and should only be used to sign messages generated internally by a TPM, which *includes* a PCR value.

Secure Processor Using TPM alone is *not sufficient* to provide a reasonable trust in the computer behavior because it suffers from the time-of-check to time-of-use problem: successful verification of the TPM-signed PCR register value only shows that the software module was loaded correctly. However, it does not 'prove' that during the execution, the integrity of the loaded module is still intact. An adversary may be able to compromise the integrity of the loaded software module between the PCR-value checking and execution of the module.

To address this problem, we need a secure environment that allows a module, once loaded, to be protected from dynamic runtime attacks. In other words, we need an isolated execution environment, *and* the ability to prove that *a set of outputs is indeed the result of correct isolated execution of an integrity-verified module based on a set of given inputs.* In essence, the proof required to convince a verifier of such an execution is generated by signing the PCR value of the module loaded into the isolated execution environment, the input values given to this module, and the output generated from the execution of this module in the isolated execution environment. These values are signed using the TPM AIK signing key. A verifier, who knows the hash value of the executed module, the input and the output values, should be able to verify the correctness of the given proof by verifying the signature using the corresponding AIK's signature verification key. We call this property *provable isolated execution.* For an example, see (McCune et al. 2008, Section 4.4).

This capability normally works *in conjunction* with a TPM, and has already been provided by most of the current processors, such as AMD's Secure Virtual Machine, Intel's TXT Technology (Intel 2007), and Flicker (McCune et al. 2008). We call such a TPM platform, which works with secure processor to provide the *provable isolated execution* property, an *extended TPM* platform.

Smart-card and TPM A smartcard has many similarities to a TPM: it can store cryptographic keys, user credentials, perform cryptographic operations, as well as executing applets. Despite these similarities, some features of a TPM are not available in a smart card. In this paper, we require a system that can verify the integrity of few security-critical modules satisfying the *provable isolated execution* property. Such a feature, while *theoretically* possible on smart card, has only been provided by an *extended TPM* platform.

Nevertheless, smart card and TPM should not be seen as competing technologies. Instead, they are complementary (Dell 2004, George 2004). There could be a role that a smart card could play in enhancing the security and, possibly, efficiency of an FSSO-PIEMC system. However, it is beyond the intended scope of this paper to provide such details.