Lecture Note: Pointcheval-Sanders scheme

CS-523

Spring 2021

This lecture note summarizes the details of the Pointcheval-Sanders (PS) attribute-based credentials (ABC) scheme.

1 Preliminaries

The PS schemes uses a type-3 bilinear group setting.[1] In this setting, there are three cyclic groups G_1, G_2, G_T, each of prime order p, generated by respectively g_1, g_2, g_T. We use multiplicative notation for these groups. For example, we write $g_1 \cdot g_1 = g_1^2 \in G_1$.

Bilinear groups moreover come with a bilinear map $e : G_1 \times G_2 \to G_T$, that satisfies the following properties:

1. *Non-degeneracy.* $e(g_1, g_2)$ generates G_T.
2. *Bilinearity.* For all $a, b \in \mathbb{Z}_p$ we have that $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$.
3. *Efficiency.* The pairing e can be efficiently computed.

Finally, for a type-3 bilinear group setting, there should be no efficiently computable isomorphisms between the groups G_1 and G_2. This property only matters for security of the scheme.

2 The Signature Scheme

The basis of the PS ABC scheme is the PS signature scheme. In a traditional signature scheme, the signer signs a single message. In the PS signature scheme, the signer signs a *vector* of L messages. We introduce this signature scheme first.

- $\mathsf{KeyGen}(L)$. The key generation algorithm is run by the signer and takes as input the length L of the message vector. The signer picks $x, y_1, \ldots, y_L \in_R \mathbb{Z}_p$ uniformly at random. It also pick random generators $g \in_R G_1$ and $\tilde{g} \in_R G_2$. It then computes:

$$X = g^x,$$
$$\tilde{X} = \tilde{g}^x$$

[1] There are other bilinear group settings, such as type-1, type-2, and type-4, but type-3 is the most used in practice, so we will focus on this here.

as well as

$$Y_i = g^{y_i},$$
$$\tilde{Y}_i = \tilde{g}^{y_i}$$

for $i = 1, \ldots, L$. It outputs the public key $pk = (g, Y_1, \ldots, Y_L, , \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_L)$ as well as the private key $sk = (x, X, y_1, \ldots, y_L)$.[2]

- $\mathsf{Sign}(sk, m_1, \ldots, m_L)$. The signing algorithm is run by the signer and takes as input the private key $sk = (x, X, y_1, \ldots, y_L)$ as well as the message vector $(m_1, \ldots, m_L)$. The signer picks a random generator $h \in_R G_1^*$ and returns the signature
$$\sigma = (h, h^{x+\sum_{i=1}^{L} y_i m_i}).$$

- $\mathsf{Verify}(pk, \sigma, m_1, \ldots, m_L)$. Everybody can verify that a signature $\sigma = (\sigma_1, \sigma_2)$ over a message vector $(m_1, \ldots, m_L)$ is valid with respect to a public key $pk = (g, Y_1, \ldots, Y_L, , \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_L)$. To do so, check that $\sigma_1 \neq 1_{G_1}$ (e.g., that σ_1 is not the unity element in G_1) and that
$$e(\sigma_1, \tilde{X} \prod_{i=1}^{L} \tilde{Y}_i^{m_i}) = e(\sigma_2, \tilde{g}).$$
If both checks pass, return $\top$, otherwise return $\bot$.

3 Attribute-Based Credentials

Credentials in the Pointcheval-Sanders scheme are simply PS signatures. However, to create a true attribute-based credential scheme, we must design two protocols: an issuance protocol to obtain a credential, and a showing protocol to prove possession of a valid credential over a set of attributes. In the following, we assume that the issuer already ran KeyGen to generate the public-private key pair (pk, sk).

Issuance protocol The issuance protocol in an ABC scheme let's the issuer (acting as the signer) provide a credential to a user without having to know all the attributes it is signing. The corresponding issuance protocol proceeds as follows between a user and the issuer:

1. *Common input.* The user and issuer agree on the set of attribute indices $\mathcal{I} \subset \{1, \ldots, L\}$ that are determined by the issuer and the set of attribute indices $\mathcal{U} \subset \{1, \ldots, L\}$ that are determined by the user. Together $\mathcal{I} \cup \mathcal{U} = \{1, \ldots, L\}$.

2. *User input.* The user takes as input the issuer's public key $pk = (g, Y_1, \ldots, Y_L, , \tilde{g}, \tilde{X}, \tilde{Y}_1, \ldots, \tilde{Y}_L)$ as well as attributes values a_i for $i \in \mathcal{U}$.

[2] The values $g, Y_1, \ldots, Y_L$ in the public key and the value X in the private key are not used for the signature scheme. But we need them later when using the signature scheme to construct an ABC scheme.

3. *Issuer input.* The issuer takes as input its private key $sk = (x, X, y_1, \dots, y_L)$ and public key $pk = (g, Y_1, \dots, Y_L, , \tilde{g}, \tilde{X}, \tilde{Y}_1, \dots, \tilde{Y}_L)$ as well as attributes values a_i for $i \in \mathcal{I}$ for attributes determined by the issuer.

4. *User commitment.* First, the user commits to the attributes they want to include in the credential by picking $t \in_R \mathbb{Z}_p$ at random and computing:

   $$C = g^t \prod_{i \in \mathcal{U}} Y_i^{a_i}$$

   as well as a non-interactive proof π that C has been computed correctly

   $$\pi = \mathrm{PK}\{(t, (a_i)_{i \in \mathcal{U}}) \ : \ C = g^t \prod_{i \in \mathcal{U}} Y_i^{a_i}\}.$$

   (This proof can for example be constructed by applying the Fiat-Shamir heuristic to the corresponding Sigma protocol.) The user sends (C, π) to the issuer.

5. *Issuer signing.* The issuer verifies the validity of proof π with respect to commitment C and aborts if the proof is not correct. Otherwise, the issuer proceeds as follows. It picks $u \in_R \mathbb{Z}_p$ at random and sets

   $$\sigma' = \left(g^u, \left(XC \prod_{i \in \mathcal{I}} Y_i^{a_i} \right)^u \right).$$

   The issuer sends σ' as well as the values of the attributes chosen by the issuer, a_i for $i \in \mathcal{I}$, to the user.

6. *Unblinding signature.* The user receives the signature $\sigma' = (\sigma'_1, \sigma'_2)$ as well as the attributes and computes the final signature:

   $$\sigma = \left(\sigma'_1, \frac{\sigma'_2}{(\sigma'_1)^t} \right). \tag{1}$$

   If σ is a valid PS signature on the attributes $a_1, \dots, a_L$ the user stores the signature σ and the attributes $a_1, \dots, a_L$ as their credential.

Showing protocol. Users in possession of a credential over a set of attributes can use the showing protocol to convince a verifier that they possess such an credential. In the PS ABC scheme, the user proves possession of a valid credential using a zero-knowledge proof. As part of this proof, the user can choose to reveal some of the attribute values, while hiding others from the verifier. The showing protocol proceeds as follows between a user and a verifier.

1. *Common input.* The user and verifier agree on the public key $pk = (g, Y_1, \dots, Y_L, , \tilde{g}, \tilde{X}, \tilde{Y}_1, \dots, \tilde{Y}_L)$ of the issuer and the set of attributes $\mathcal{D}$ that should be disclosed to the verifier. Let $\mathcal{H} = \{1, \dots, L\} \setminus \mathcal{D}$ be the set of attributes that are hidden from the verifier.

2. *Creating disclosure proof.* The user takes as input a signature $\sigma = (\sigma_1, \sigma_2)$ over the attributes $a_1, \dots, a_L$ as input. They create the disclosure proof as follows:

(a) The user picks random values $r, t \in_R \mathbb{Z}_p$ and computes a randomized signature $\sigma' = (\sigma'_1, \sigma'_2) = (\sigma_1^r, (\sigma_2 \sigma_1^t)^r)$.

(b) The user computes a (non-interactive) zero-knowledge proof that the signature σ' is a valid signature:

$$\pi = \mathrm{PK}\Bigg\{ (t, (a_i)_{i \in \mathcal{H}}) : \\ \frac{e(\sigma'_2, \tilde{g}) \cdot \prod_{i \in \mathcal{D}} e(\sigma'_1, \tilde{Y}_i)^{-a_i}}{e(\sigma'_1, \tilde{X})} = e(\sigma'_1, \tilde{g})^t \prod_{i \in \mathcal{H}} e(\sigma'_1, \tilde{Y}_i)^{a_i} \Bigg\}.$$

Finally, the user sends the randomized signature σ', the disclosed attributes $(a_i)_{i \in \mathcal{D}}$, and the proof π to the verifier.

3. *Verify a disclosure proof.* The verifier receives the signature $\sigma' = (\sigma'_1, \sigma'_2)$, the disclosed attributes $(a_i)_{i \in \mathcal{D}}$, and the proof π from the user. The verifier then:

(a) Verifies that $\sigma_1 \neq 1_{G_1}$, i.e., that σ_1 is not the unity element in G_1.

(b) Checks that the user has a valid signature under public key *pk* over the disclosed attributes $(a_i)_{i \in \mathcal{D}}$ by verifying the proof π.

If both checks succeed, the verifier accepts the disclosure proof.