On implementing Pairing-Based Protocols (on ordinary curves)

A work almost finished

(don't read the authors' list if you are a reviewer)

Eric Zavattoni, Luis J. Dominguez Perez, Shigeo Mitsunari, Ana H Sanchez-Ramirez, Tadanori Teruya, and Francisco Rodriguez-Henriquez.

ldominguez@tamps.cinvestav.mx

ECC 2012, rump session

This talk is also known as: Attribute Based Cryptography: Type 3 pairing with attributes in $\mathbb{G}_1$ vs. $\mathbb{G}_2$.

Having the fastest pairing function implemented with all the bells and whistles is impressive, however, sometimes we need to go further and think about the ones who design pairing protocols.

Having the fastest pairing function implemented with all the bells and whistles is impressive, however, sometimes we need to go further and think about the ones who design pairing protocols.

These guys do amazing things, but also the ones that implement the protocols *may come with a slightly different requirements... sometimes anyway*

Building blocks for PB Protocols

To implement a protocol of this type (on ORDINARY CURVES), we need:

- A hash function, $\oplus$
- Hash into $\mathbb{G}_1$
- Hash into $\mathbb{G}_2$
- Exponentiation in $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$
- Point addition/doubling in $\mathbb{G}_1$ and $\mathbb{G}_2$
- Exponentiation in $\mathbb{F}_{p^k}$?
- LSSS
- Multipairing
- Fixed parameter pairing

There are a lot of work already on the **hash function**, in deed, recently, NIST has chosen the SHA-3 winner

There are a lot of work already on the **hash function**, in deed, recently, NIST has chosen the SHA-3 winner

Hash into $\mathbb{G}1$**.** There is now a method to hash in deterministic time or some p.f.e. curves.

There are a lot of work already on the **hash function**, in deed, recently, NIST has chosen the SHA-3 winner

Hash into $\mathbb{G}1$. There is now a method to hash in deterministic time or some p.f.e. curves.

Hash into $\mathbb{G}2$. We have now the Fuentes et al. method

There are a lot of work already on the **hash function**, in deed, recently, NIST has chosen the SHA-3 winner

Hash into $\mathbb{G}1$**.** There is now a method to hash in deterministic time or some p.f.e. curves.

Hash into $\mathbb{G}2$**.** We have now the Fuentes et al. method

Exponentiation in $\mathbb{G}_1$**.** We use the GLV method

There are a lot of work already on the **hash function**, in deed, recently, NIST has chosen the SHA-3 winner

Hash into $\mathbb{G}1$. There is now a method to hash in deterministic time or some p.f.e. curves.

Hash into $\mathbb{G}2$. We have now the Fuentes et al. method

Exponentiation in $\mathbb{G}_1$. We use the GLV method

Exponentiation in $\mathbb{G}_2$, $\mathbb{G}_T$. We use the GS method

There are a lot of work already on the **hash function**, in deed, recently, NIST has chosen the SHA-3 winner

Hash into $\mathbb{G}1$. There is now a method to hash in deterministic time or some p.f.e. curves.

Hash into $\mathbb{G}2$. We have now the Fuentes et al. method

Exponentiation in $\mathbb{G}_1$. We use the GLV method

Exponentiation in $\mathbb{G}_2$, $\mathbb{G}_T$. We use the GS method (no, the Karabina's exponentiation is not feasible here)...

There are a lot of work already on the **hash function**, in deed, recently, NIST has chosen the SHA-3 winner

Hash into $\mathbb{G}1$. There is now a method to hash in deterministic time or some p.f.e. curves.

Hash into $\mathbb{G}2$. We have now the Fuentes et al. method

Exponentiation in $\mathbb{G}_1$. We use the GLV method

Exponentiation in $\mathbb{G}_2$, $\mathbb{G}_T$. We use the GS method (no, the Karabina's exponentiation is not feasible here)... for a single core ;)

EC arithmetic. Have a look at the Explicit Database

EC arithmetic. Have a look at the Explicit Database

LSSS. To break a secret into several parts, we use the Liu and Cao method to transfer a linear secret-sharing scheme matrix into a cyphertext-policy Attribute-Based Encryption.

Multipairing. (Product of pairings) As presented in the tutorial section by Francisco, we can share the accummulator.

Multipairing. (Product of pairings) As presented in the tutorial section by Francisco, we can share the accummulator.

Fixed-parameter pairing computation. In the fixed parameter setting, the "right-hand" parameter of the curve is known in advance, hance, all of the lines computations can be precomputed in advance. The remaining thing is to perform a few multiplications in $\mathbb{F}_p$, and some simultaneous inversions.

CPABE in $\mathbb{G}_2$ (people's imp.)

Setup. Select random points $P \in \mathbb{G}_1[r]$, and $Q \in \mathbb{G}_2[r]$. Pick up random group elements α, $\delta \in \mathbb{Z}_r$. Set $Q_\alpha = [\alpha]Q$ and $Q_\delta = [\delta]Q$. Compute $v = e(P, Q)^\alpha$. Choose a hash function $H_1(\cdot)$ which hashes an attribute string to a group element $\in \mathbb{G}_2$. The public parameters are $\{P, Q, Q_\delta, v\}, \{\mathcal{H}_1, \dots \mathcal{H}_n\}$, and the Master key is $\{Q_\alpha\}$.

KeyGen. Pick up random group element $t \in \mathbb{Z}_r$. Set $K = P_\alpha + [t]P_\delta$, and $L \leftarrow [t]Q$. For all i in each of the attributes for the entity's set of attributes $\mathcal{H}$, set $K_i \leftarrow [t]\mathcal{H}_i$. The secret key is $\mathrm{SK} = (K, L, \forall i \in \mathcal{H} : K_i)$.

Encryption. Hide the secret message M in v, and a master random secret s in $C = Mv^s$, $C_d = [s]P$. For all of the attributes in the policy hide the randomly generated vectors as $C_i \leftarrow [\lambda_i]Q_\delta - [x_i]\mathcal{H}_i$ and $D_i \leftarrow [x_i]P$. The cypher text is $C_T = \mathcal{S}, C, C_d, \forall i \in [1 \ldots m] : C_i, D_i$.

Decryption. We get a vector $\bar{\omega}$. To recover the hidden message, we compute a product of pairings:

$$M = C \cdot \left(e(-[\Delta]K, C_d) \cdot e(L, \sum_{i \in \tilde{\mathcal{H}}} [\omega_i]C_i) \cdot \prod_{i \in \tilde{\mathcal{H}}} e(K_i, [\omega_i]D_i) \right)^{\frac{1}{\Delta}}.$$

CPABE in $\mathbb{G}_1$ (our imp.)

Setup. Select random points $P \in \mathbb{G}_1[r]$, and $Q \in \mathbb{G}_2[r]$. Pick up random group elements α, $\delta \in \mathbb{Z}_r$. Set $P_\alpha = [\alpha]P$ and $P_\delta = [\delta]P$. Compute $v = e(P, Q)^\alpha$. Choose a hash function $H_1(\cdot)$ which hashes an attribute string to a group element $\in \mathbb{G}_1$. The public parameters are $\{P, Q, P_\delta, v\}, \{\mathcal{H}_1, \ldots \mathcal{H}_n\}$, and the Master key is $\{P_\alpha\}$.

KeyGen. Pick up random group element $t \in \mathbb{Z}_r$. Set $K = P_\alpha + [t]P_\delta$, and $L \leftarrow [t]Q$. For all i in each of the attributes for the entity's set of attributes $\mathcal{H}$, set $K_i \leftarrow [t]\mathcal{H}_i$. The secret key is $\text{SK} = (K, L, \forall i \in \mathcal{H} : K_i)$.

Encryption. Pick a master random secret $s \in \mathbb{Z}_r$, and n random secrets $y_i \in \mathbb{Z}_r$, and form a vector $\bar{\mathbf{u}} = (s, y_2, \ldots, y_n)$. Compute a vector $\bar{\lambda} = \mathcal{S}\bar{\mathbf{u}}^T$. Pick n random secrets $\in \mathbb{Z}_r$ and form a vector $\bar{\mathbf{x}} = (x_1, \ldots, x_n)$. Hide the secret message M in v, and the master random secret in C_d. For all of the attributes in the policy hide the randomly generated vectors as $C_i \leftarrow [\lambda_i]P_d - [x_i]\mathcal{H}_i$ and $D_i \leftarrow [x_i]Q$. The cypher text is $C_T = \mathcal{S}, C, C_d, \forall i \in [1 \ldots m] : C_i, D_i$.

Decrypt.

Require: C_T, SK

Ensure: M (if the attributes in SK satisfy the policy of the C_T)

Let $\mathcal{H}'$ be the set of attributes in the policy $\mathcal{S}$ and in the SK
Let $\mathcal{S}'$ be a LSSS matrix $(m' \times n)$
$\mathcal{S}' \leftarrow$ Reduce the LSSS matrix $\mathcal{S}$ in C_T by removing the rows corresponding to an attribute not in SK
Calculate the vector $\bar{\omega}$ such that $s = \bar{\omega}.\bar{\lambda}$.
$\Delta \leftarrow \frac{Det(\mathcal{S}')}{GCD(Det(\mathcal{S}'),\bar{\omega})}$
for $i = 1 \ldots m'$ **do**
  $C_i^{\omega_i} \leftarrow [\omega_i]C_i$ {Scalar-point multiplication in $\mathbb{G}_1$}
  $K_i^{\omega_i} \leftarrow [\omega_i]K_i$ {Scalar-point multiplication in $\mathbb{G}_1$}
end for

$$M = C \cdot \left(e(C_d, -[\Delta]K) \cdot e(L, \sum_{i \in \mathcal{H}'} C_i^{\omega_i}) \cdot \prod_{i \in \mathcal{H}'} e(D_i, K_i^{\omega_i}) \right)^{\frac{1}{\Delta}}$$

{Scalar-point multiplication in $\mathbb{G}_1$, Point addition in $\mathbb{G}_1$, Multiplication $\in \mathbb{G}_T$, Multipairing, Inversion $\in \mathbb{G}_T$}

return M

$\mathbb{G}_1$

Step	$\mathbb{G}_1$			$\mathbb{G}_2$			$\mathbb{G}_T$		Pairing	
	S.M.U.	S.M.C.	s.S.M.	S.M.U.	S.M.C.	s.S.M.	E.U.	E.C.	U.	K.
Encrypt:	–	2n	–	–	n+1	–	–	1	–	–
Keygen:	–	n+1	–	–	1	–	–	–	–	–
Decrypt $\Delta = 1$:	–	–	2n	–	–	–	–	–	n+2	–
Decrypt $\Delta \neq 1$:	–	–	2n	–	–	–	–	1	n+2	–

$\mathbb{G}_2$

Step	$\mathbb{G}_1$			$\mathbb{G}_2$			$\mathbb{G}_T$		Pairing	
	S.M.U.	S.M.C.	s.S.M.	S.M.U.	S.M.C.	s.S.M.	E.U.	E.C.	U.	K.
Encrypt:	–	n+1	–	–	2n	–	–	1	–	–
Keygen:	–	1	–	–	n+1	–	–	–	–	–
Decrypt $\Delta = 1$:	–	–	n	–	–	n	–	–	1	n+2
Decrypt $\Delta \neq 1$:	–	–	n	–	–	n	–	1	1	n+2

Our results.

LSSS ABE Protocole	CPU cycles			
	Our results $\mathbb{G}_1$		Our estimates $\mathbb{G}_2$	
	Six attributes	Twenty attributes	Six attributes	Twenty attributes
Encrypt	3 142 K	9 357 K	3 782K	11 787K
Keygen	997 K	2 711 K	1 764K	5 260K
Decrypt ($\Delta = 1$)	6 441 K	17 992 K	4 044K	11 392K

LSSS ABE Protocole	CPU cycles Scott results $\mathbb{G}_2$ Six attributes	 Twenty attributes
Encrypt	16 704 K	31 320 K
Keygen	3 408 K	–
Decrypt ($\Delta = 1$)	14 832 K	23 8320 K