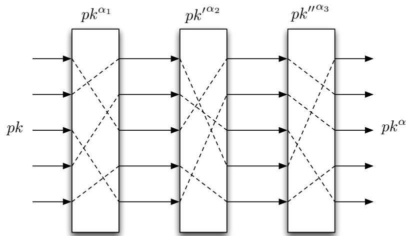
UniCrypt 2.0:
Proof Generator and Mixer

Philipp Locher

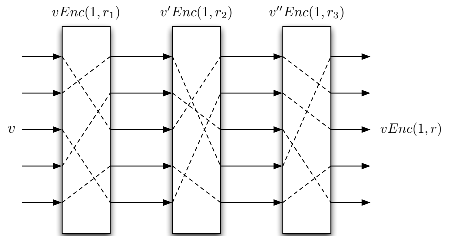
Seminar, E-Voting Group, BFH

March 25, 2014

Motivation

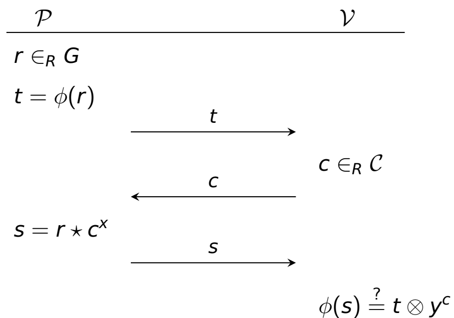


Identity Mixer



Re-Encryption Mixer

Preimage Proof



Preimage Proof

$\mathcal{P}$		$\mathcal{V}$
$r \in_R G$		
$t = \phi(r)$		
$c = H(y, t)$		
$s = r \star c^x$		
	$\xrightarrow{s}$	
		$\phi(s) \stackrel{?}{=} t \otimes y^c$

Challenge

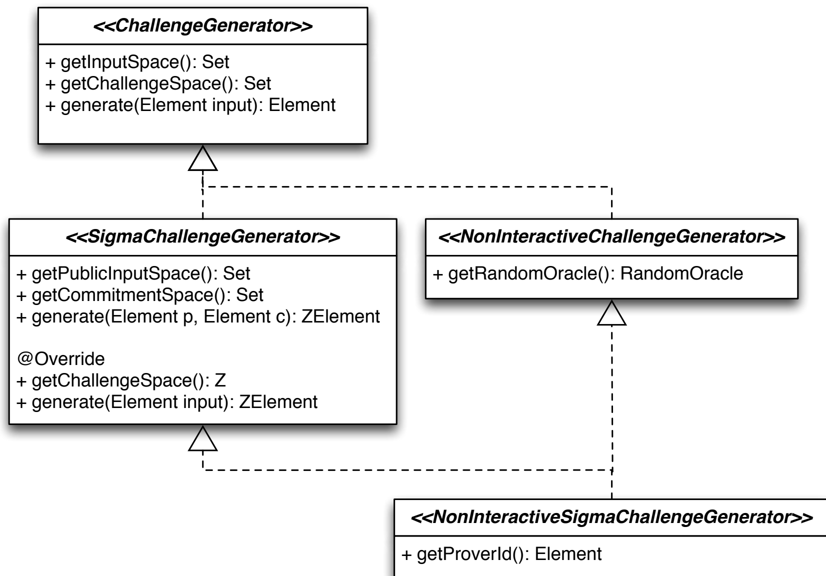
Who is responsible for the challenge?

Challenge

Who is responsible for the challenge?

The challenge generator!

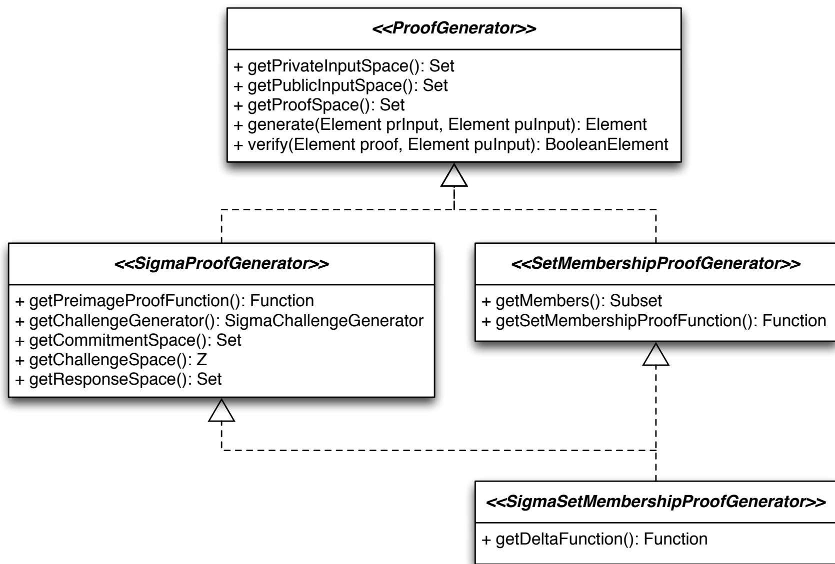
Challenge Generator



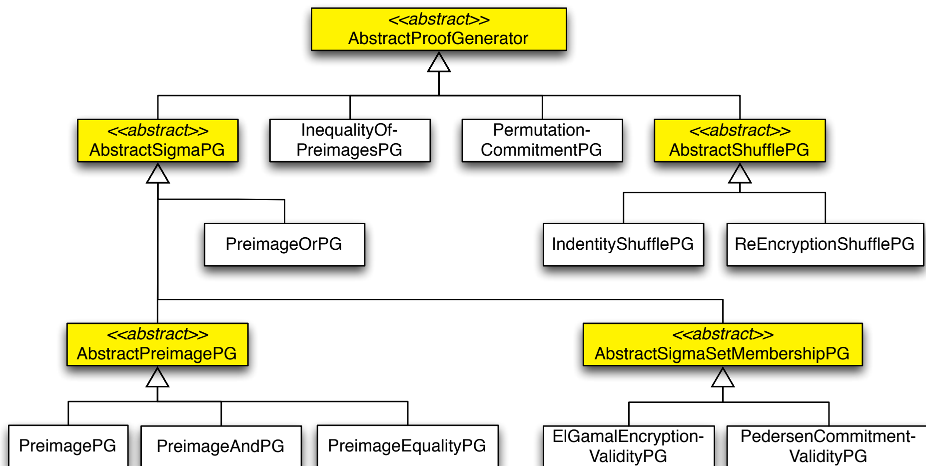
Challenge Generator

```
CyclicGroup G_q;
StringMonoid sm;

ChallengeGenerator cg =
    StandardNonInteractiveChallengeGenerator.getInstance(sm, G_q, 10);

Tuple challenges = (Tuple) cg.generate(sm.getElement("input X"));

//----------------------


Function f;

SigmaChallengeGenerator scg =
    StandardNonInteractiveSigmaChallengeGenerator.getInstance(f, proverId);

ZElement challenge = scg.generate(publicInput, commitment);
```

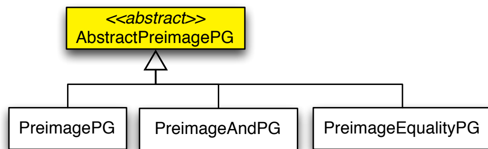
Proof Generator



Proof Generator



Preimage Proof Generator

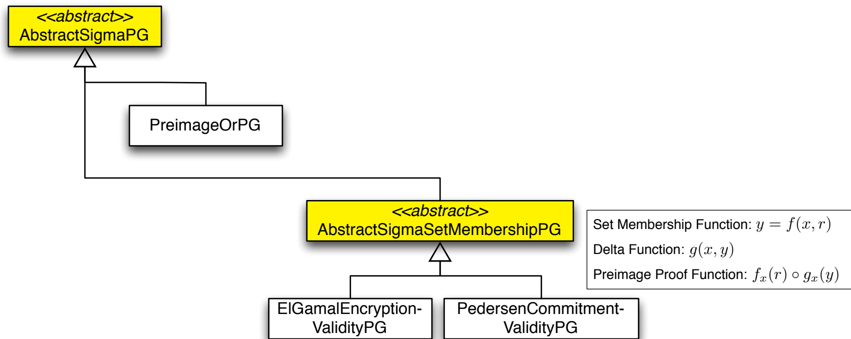


- Private/Public input corresponds to the domain/co-domain of the proof function

Preimage Proof Generator

```
// f: Z_q -> G_q
//    y = f(x) = 4^x

CyclicGroup G_q;
Function f = GeneratorFunction.getInstance(G_q.getElement(4));

SigmaChallengeGenerator scg =
    StandardNonInteractiveSigmaChallengeGenerator.getInstance(f, proverId);
PreimageProofGenerator pg =
    PreimageProofGenerator.getInstance(scg, f);

Element privateInput = G_q.getZModOrder().getElement(3);
Element publicInput = G_q.getElement(64);

Triple proof = pg.generate(privateInput, publicInput);
BooleanElement v = pg.verify(proof, publicInput);
```

Sigma Proof Generator

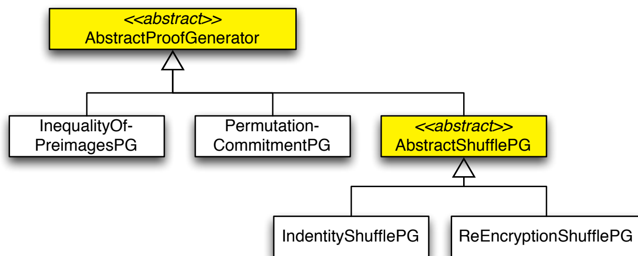


- Private/Public input doesn't correspond to the domain/co-domain of the proof function
- The proof function can be built based on common values

Sigma Proof Generator

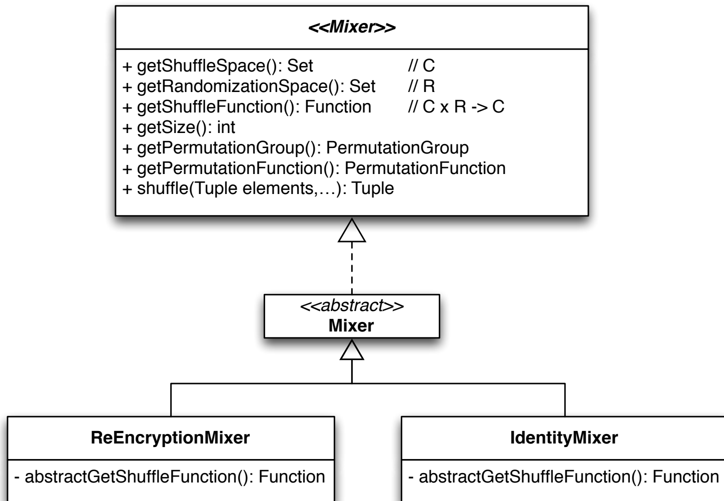
```
// ElGamal Encryption Validity Proof
// Plaintexts: {4, 2, 8, 16}, g = 2, pk = 4
// Valid tuple: (2^3, 4^3*2) = (8, 128)

CyclicGroup G_q;
ElGamalEncryptionScheme elGamalES =
    ElGamalEncryptionScheme.getInstance(G_q.getElement(2));
Element publicKey = G_q.getElement(4);

Subset plaintexts = Subset.getInstance(G_q, new Element[]{
    G_q.getElement(4), G_q.getElement(2),...});


ElGamalEncryptionValidityProofGenerator pg =
    ElGamalEncryptionValidityProofGenerator
        .getInstance(elGamalES, publicKey, plaintexts, proverId);

Pair publicInput = Pair.getInstance(G_q.getElement(8), G_q.getElement(128));

Element secret = G_q.getZModOrder().getElement(3);
int index = 1;
Pair privateInput = pg.createPrivateInput(secret, index);

Triple proof = pg.generate(privateInput, publicInput);
BooleanElement v = pg.verify(proof, publicInput);
```

Proof Generator



- The proof function is based on public input
- A `PreimageProofGenerator` might be used internally

Mixer

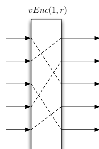


Mixer

```
// ElGamal Encryptions Mixer

CyclicGroup G_q;
Tuple ciphertexts;
int size = ciphertexts.getArity();

ElGamalEncryptionScheme elGamalES =
    ElGamalEncryptionScheme.getInstance(G_q.getElement(2));
Element publicKey = G_q.getElement(4);


ReEncryptionMixer mixer =
    ReEncryptionMixer.getInstance(elGamalES, publicKey, size);

PermutationElement permutation =
    mixer.getPermutationGroup().getRandomElement();

Tuple randomizations = mixer.generateRandomizations();

Tuple sCiphertexts = mixer.shuffle(ciphertexts, permutation, randomizations);
```

Finally...

- Shuffle



- Permutation Commitment Proof

$$\Sigma\text{-proof}\left[\begin{matrix} v, w \in \mathbb{Z}_q \\ \bar{e}' \in \mathbb{Z}_q^N \end{matrix} \;\middle|\; Com(\bar{1}, v) = c_\pi{}^{\bar{1}} \wedge Com(\bar{e}', w) = c_\pi{}^{\bar{e}} \wedge \prod_{i=1}^{N} e_i' = \prod_{i=1}^{N} e_i \right]$$

- Shuffle Proof

$$\Sigma\text{-proof}\left[\begin{matrix} r, w \in \mathbb{Z}_q \\ \bar{e}' \in \mathbb{Z}_q^N \end{matrix} \;\middle|\; Com(\bar{e}', w) = c_\pi{}^{\bar{e}} \wedge \prod_{i=1}^{N} (u_i')^{e_i'} = ReEnc(\prod_{i=1}^{N} (u_i)^{e_i}, r) \right]$$