PAILLIER ZERO-KNOWLEDGE PROOF

TAYLOR FOX DAHLIN, DAYLIGHTING SOCIETY

December 17, 2016

Abstract

Suppose two parties are communicating, with a third party passing the messages from one party to the other. However, this third party will only allow the message through if it deems the message acceptable, presumably from a pre-approved list of messages. Using the Paillier cryptosystem, it is possible for this third party to determine whether the encrypted message being passed is a member of this pre-approved list, without ever seeing the actual contents of the message, nor knowing which message it is. The process for doing this is detailed below.

CONTENTS

1 INTRODUCTION

Let c be the Paillier encryption of your message m using random value r and public key (g, n) such that:

$$c = g^m * r^n \mod n^2$$

Furthermore, let $\mathbb{Z}_n^*$ be the set of integers coprime to n, and let b be the preselected bit-string length used during the proof.

Suppose you have K valid messages,

$$m_1, m_2, m_3, ..., m_k$$

and this set contains your message m. In order for an external party who does not know m or r determine whether c is an encryption of some m_i, the following zero-knowledge proof can be performed. Using these valid messages, you can calculate the following set of values,

$$u_1, u_2, u_3, ..., u_k$$

using the following formula:

$$u_i = \frac{c}{g^{m_i}} \mod n^2$$

This is useful for our calculations because of the special property where $m_i = m$:

$$u_i = \frac{c}{g^{m_i}} = \frac{g^m * r^n}{g^m} = r^n \mod n^2$$

2 PREPARATION

First, the prover randomly selects a set of values,

$$e_1, e_2, e_3, ..., e_k \in 2^b < min(p, q); z_1, z_2, z_3, ..., z_k \in \mathbb{Z}_n^*$$

without selecting e_m or z_m, as these require calculation to determine. Next, the prover randomly selects $\omega \in Z_n^*$.

3 COMMITMENT

Prover calculates the set of values,

$$a_1, a_2, a_3, ..., a_k$$

such that for all a_i (except where $m_i = m$), this equation holds:

$$a_i = \frac{z_i^n}{u_i^{e_i}} \mod n^2$$

For $m_i = m$, we calculate a_i as follows:

$$a_i = \omega^n \mod n^2$$

The prover then sends $a_1, a_2, a_3, ..., a_k$ to the verifier.

4 CHALLENGE

At this point of the zero-knowledge proof there are two ways to proceed, at the discretion of the implementer: the proof can be be interactive, or non-interactive. Both implementations generate some bit-string e, which is used by the prover in further calculations for their proof. The probability of forgery given bit-string e with length b_e is:

$$P(forgery) = \frac{1}{2^{b_e}}$$

4.1 *Interactive*

Upon receiving the commitment from the prover, the verifier can randomly generate a bit-string e of length b and send this to the prover.

4.2 *Non-Interactive*

In a non-interactive implementation, the hash of the commitment is used for the bit-string e. A secure hashing algorithm ensures that crafting the hash collisions is exceptionally unlikely, which allows the prover to use this in place of further interactions with the verifier. This is the method used in the Ruby gem made available by the Daylighting Society.*

*See `https://paillier.daylightingsociety.org`

5 RESPONSE

The prover now calculates z_i and e_i for $m_i = m$ as follows:

$$e_i = e_{challenge} - \sum_{k=1}^{K} e_k$$

$$z_i = \omega * r^{e_i} \mod n$$

After performing these calculations, the prover sends the sets of values,

$$e_1, e_2, e_3, ..., e_k; z_1, z_2, z_3, ..., z_k$$

to the verifier, including the newly calculated e_i and z_i in their appropriate positions in these sets.

6 VERIFICATION

The verifier now has all a_i, e_i, and z_i from the prover, and can determine whether the prover performed their calculations correctly. First, the verifier confirms the following:

$$\sum_{k=1}^{K} e_k = e_{challenger} \mod 2^b$$

If this fails, then the prover did not follow the rules or attempted to cheat.
Next, the verifier confirms the following:

$$z_k^n = a_k * u_k^{e_k} \mod n^2$$

For all k such that $m_k \neq m$, this is guaranteed as it is pre-calculated by the prover. For k such that $m_k = m$, this calculation works as follows:
Recall the following:

$$u_i = r^n \mod n^2$$

$$z_i = \omega * r^{e_i} \mod n$$

$$a_i = \omega^n \mod n^2$$

With this in mind, we can substitute each element in the verification equation:

$$z_k^n = a_k * u_k^{e_k} \mod n^2$$

$$(\omega * r^{e_i})^n = \omega^n * (r^n)^{e_k} = \omega^n * (r^{e_k})^n = (\omega * r^{e_i})^n \mod n^2$$

If this fails, then the prover did not follow the rules or attempted to cheat.

If both tests have passed however, the verifier now has proof that the encrypted message is from the set of valid messages, having never seen the plaintext, and without being able to solve for the plaintext.