Discovering Cryptography further: Elgamal

Genalyn Estrada (745720) & Kevin Norman (787037), *Swansea University, Computer Science Department*

Elgamal is an asymmetric key encryption algorithm for public-key cryptography which is based on the Diffie–Hellman (D-H) key exchange. It was described by Taher Elgamal in 1985 [3]. The cryptographic protection offered by the Elgamal algorithm depends on the difficulty of computing discrete logs in a large prime modulus [6].

A Brief Introduction

Elgamal simplifies the Diffie-Hellman key exchange algorithm by introducing a random exponent. This random exponent is a replacement for the private exponent of the receiving entity under the DH algorithm. Due to this simplification, the algorithm can be used to encrypt in one direction, without the necessity of the second party to take an active role.

Elgamal uses randomization (through choosing an exponent) which means that each plaintext message has many different possible ciphertexts, thus it is considered a probabilistic encryption scheme. However, due to this randomization, messages encrypted with Elgamal creates ciphertext twice as big as the original message. This is a common problem with asymmetric cryptography, and explains why it is mainly used to exchange a key for synchronous cryptography [7].

The Phases of Elgamal

Set-up Phase

The set-up phase is done by the party interested in receiving messages. This is only done *once* and is published publicly. The prime number chosen needs to be at least 1024 bits. This provides a security roughly equivalent to an 80-bit symmetric cipher key. For better long-term security, a prime of length 2048 bits or greater should be chosen [9].

1. Bob chooses a large prime ρ and primitive root of the prime number α which will be used as a generator: $\alpha \in \mathbb{Z}_{\rho}^{*}$.
2. Bob chooses a random number between $\{2, 3, \ldots \rho - 2\}$ and this becomes his private key which is $\kappa_{pr} = \delta \in \{2, 3, \ldots \rho - 2\}$
3. Bob computes his public key by computing the generator raised to the random number that he chose, and everything is encompassed by mod p, which is $\kappa_{pub} = \beta \equiv \alpha^{\delta} \, mod \, \rho$
4. Bob publishes (ρ, α, β) publicly.

Encryption Phase

This phase is done by the party who wants to send the message. This is done every time a message is sent.

1. Alice chooses a random number between $\{2, 3, \ldots \rho - 2\}$ which will be used to compute her public key: $\iota \in \{2, 3, \ldots \rho - 2\}$
2. Alice computes her public key which is $\kappa_E \equiv \alpha^{\iota} mod \rho$. κ_E is also known as ephemeral key, which is a temporary key. This key is different every time a message is sent.
3. Alice computes her session key which is $\kappa_M \equiv \beta^{\iota} \, mod \, \rho$. κ_M. The session key is also known as the masking key.
4. Alice creates ciphertext with the masking key: $y \equiv x \cdot \kappa_M mod \rho$
5. Alice sends (y, κ_E) to Bob.

Decryption Phase

This phase is done by the receiver, ie the person that also completed the set-up phase. This is done every time a message is received. Here, Bob will send Alice a message. It is important to note here that the public key defined in the encryption phase is just that; public. Anybody can who has Alice's public key can send Alice encrypted messages. Alice holds the private key however and as such, only she can decrypt these messages.

1. Bob computes the inverse to unmask: $\kappa_M \equiv {\kappa_E}^{\delta} mod \rho$
2. Bob gets the plaintext : $x \equiv y \cdot {\kappa_M}^{-1} mod \rho$

Example 1

Set-up Phase

1. Bob chooses a prime number $\rho = 79$ and $\alpha \in \mathbb{Z}_{79}^*$, $\alpha = 30$.
2. Bob chooses a random number between $\{2, 3, \ldots 77\}$, $\delta = 61$ as his private key.
3. Now Bob computes his public key:
   $\kappa_{pub} = \beta \equiv 30^{61} \; mod \; 79$ which is 59.
4. Bob publishes $(79, 30, 59)$ publicly (His pubkey).

Encryption Phase

Plaintext = 44.

1. Alice chooses a random number $\iota = 4$ between $\{2, 3, \ldots 77\}$.
2. Alice computes her public key which is
   $\kappa_E \equiv 30^4 \; mod \; 79$ which is 13.
3. Alice computes her session key which is
   $\kappa_M \equiv 59^4 \; mod \; 79$ which is 25.
4. Alice makes ciphertext with the masking key:
   $y \equiv 44 \cdot 25 \; mod \; 79$ which is 73.
5. Alice sends $(73, 13)$ to Bob.

Decryption Phase

1. Bob computes the inverse to unmask:
   $\kappa_M \equiv 13^{61} \; mod \; 79$ resulting in 25.
2. Bob gets the plaintext:
   $x \equiv 73 \cdot 25^{-1} \; mod \; 79$ which is 44

Example 2

Set-up Phase

1. Bob chooses a prime number $\rho = 1087$ and $\alpha \in \mathbb{Z}_{1087}^*$, $\alpha = 58$.
2. Bob chooses a random number between $\{2, 3, \ldots 1085\}$, $\delta = 32$ as his private key.
3. Now Bob computes his public key:
   $\kappa_{pub} = \beta \equiv 58^{32} \; mod \; 1087$ which is 137
4. Bob publishes $(1087, 58, 137)$ publicly.

Encryption Phase

Plaintext = 54.

1. Alice chooses a random number $\iota = 10$ between $\{2, 3, \ldots 1085\}$.
2. Alice computes her public key which is
   $\kappa_E \equiv 58^{10} \; mod \; 1087$ which is 965.
3. Alice computes her session key which is
   $\kappa_M \equiv 137^{10} \; mod \; 1087$ which is 1077.
4. Alice creates ciphertext with the masking key:
   $y \equiv 54 \cdot 1077 \; mod \; 1087$ which is 547.
5. Alice sends $(547, 965)$ to Bob.

Decryption Phase

1. Bob computes the inverse to unmask:
   $\kappa_M \equiv 965^{32} \; mod \; 1087$ resulting in 1077.
2. Bob gets the plaintext:
   $x \equiv 547 \cdot 1077^{-1} \; mod \; 1087$ which is 54

Proof of Correctness

The masking key κ_M, is substituted with the definition shown on step 10, with inverse. This proof was provided by Christof Parr in a lecture he gave on the topic [10].

$$y \cdot {\kappa_M}^{-1} \equiv y({\kappa_E}^{\delta})^{-1} \, mod \, \rho \tag{1}$$

y is substituted with the definition of y shown on step 8.

$$y \cdot {\kappa_M}^{-1} \equiv x \cdot \kappa_M \cdot \kappa_E^{-\delta} \, mod \, \rho \tag{2}$$

The masking key κ_M is substituted again, this time with the earlier definition shown on step 7. The ephemeral key, κ_E is also substituted with the value shown in step 6.

$$y \cdot {\kappa_M}^{-1} \equiv x \cdot \beta^{\iota} \cdot (\alpha^{\iota})^{-\delta} \, mod \, \rho \tag{3}$$

β^{ι} is substituted with the definition given in step 3.

$$y \cdot {\kappa_M}^{-1} \equiv x \cdot (\alpha^{\delta})^{\iota} \cdot (\alpha^{\iota})^{-\delta} \, mod \, \rho \tag{4}$$

We rewrite the equation as:

$$y \cdot {\kappa_M}^{-1} \equiv x \cdot \alpha^{\iota\delta} \cdot \alpha^{-\iota\delta} \, mod \, \rho \tag{5}$$

$$y \cdot {\kappa_M}^{-1} \equiv x \cdot \alpha^{\iota\delta - \iota\delta} \, mod \, \rho \tag{6}$$

$$y \cdot {\kappa_M}^{-1} \equiv x \cdot \alpha^{0} \, mod \, \rho \tag{7}$$

Anything raised to 0 will equal 1 therefore:

$$y \cdot {\kappa_M}^{-1} \equiv x \cdot 1 \, mod \, \rho \tag{8}$$

$x \cdot 1 = x$ therefore:

$$y \cdot {\kappa_M}^{-1} \equiv x \, mod \, \rho \tag{9}$$

Security Assessment

In order to assess the security of Elgamal, we must distinguish between passive, (listen-only) and active attacks (generating messages).

Passive

Recovering the plaintext x from the information passed $\rho, \alpha, \beta = \alpha^{\delta}, \kappa_E = \alpha^{\iota}$ and $y = x \cdot \beta^{\iota}$ which is obtained by eavesdropping, relies on the hardness of the DiffieHellman problem [9]. Currently there is no other method known for solving the DHP than computing discrete logarithms. If we assume that there was somehow a way to compute DLPs, there are two ways of attacking the Elgamal scheme:

Recover x by Finding Bobs Secret Key

$$\delta = \log_{\alpha} \beta mod \rho \tag{10}$$

This step solves the DLP, which as described before is *computationally infeasible* if the parameters are chosen correctly (ie. ρ should at least have a length of 1024 bits). However, if Eve succeeds, she can decrypt the plaintext by performing the same steps as described in the decryption phase.

Recover x by finding Alice's random exponent

$$\iota = \log_{\alpha} \kappa mod \rho \tag{11}$$

Once again, this step solves the DLP. If Eve succeeds, she can decrypt the plaintext by performing $x \equiv y \cdot (\beta^{\iota})^{-1} mod \rho$

Active

Public Key Authenticity

As Elgamal is an asymmetric scheme, we must be ensure that the public keys are authentic. There is no way for Alice to know for sure that the public key belongs to Bob. This can be prevented by using certificates.

Reusing Alice's i

As stated in the phases of Elgamal, the encryption phase has to be repeated every time Alice wants to send a message. This is so that Alice's random exponent ι is not repeated. Eve can detect this repetition as the ephemeral keys would be identical. If Alice was to send two subsequent messages $x1$ and $x2$, it would result in the same masking key. This would be sent as (y_1, κ_E) and (y_1, κ_E). If Eve can compute the first message, then she can compute the masking key as $\kappa_M \equiv y_1 x_1^{-1} mod \rho$. With this knowledge, Eve can decrypt $x2$ as $x_2 \equiv y_2 \kappa_M^{-1} mod \rho$. Any other message encrypted with the same ι would be recovered in the same way. This can be countered by using a cryptographically secure Pseudorandom number generator. [8]

Replacing the ciphertext

If Eve observes that Bob receives (y, κ_E), she can replace it to be (sy, κ_E) where s is an integer. The receiver would decrypt $sx mod \rho$, the decrypted text would be a multiple of s. Eve is not able to decrypt the ciphertext, but she can manipulate it. This attack is exactly the same as attacks for RSA encryption schemes. In order to resolve this, we can follow RSA's lead by introducing padding. [2]

Comparison with RSA

RSA is a much more well known cryptographic algorithm and remains the most heavily used cryptosystem in existence. Whereas it relies on the fact that it is computationally difficult to factor large integers, Elgamal relies on the fact that it is computationally difficult to compute discrete logs in a large prime modulus [6]. Elgamal requires more computational work to encrypt data than RSA, but requires less computational work to decrypt data [1]. Elgamal does not have a formal technical specification, meaning that creating interoperable tools making use of Elgamal is difficult. RSA however has a formal technical specification; making implementing it trivial [5]. RSA was originally under patent, however Elgamal is available in the public domain - and as such software such as the GPG makes use of it [4]. Ciphertext created using Elgamal is twice the size as for RSA, for at least the same level of security [6].

References

[1] Shravya Shetty K et al Annapoorna Shetty. A review on asymmetric cryptography - rsa and elgamal algorithm. 2014.

[2] Dan Boneh. The decision diffie-hellman problem. *Algorithmic number theory*, pages 48–63, 1998.

[3] Taher ElGamal. A public key cryptosystem and a signature scheme based on discrete logarithms. *IEEE transactions on information theory*, 31(4):469–472, 1985.

[4] The Free Software Foundation. The gnu privacy handbook. 1999.

[5] RSA Laboratories. Pkcs #1 v2.2: Rsa cryptography standard. 2012.

[6] Jeffrey S. Leon. The elgamal public key encryption algorithm.

[7] Andreas V Meier. The elgamal cryptosystem, 2005.

[8] Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone. *Handbook of applied cryptography*. CRC press, 1996.

[9] Christof Paar and Jan Pelzl. *Understanding cryptography: a textbook for students and practitioners*. Springer Science & Business Media, 2009.

[10] Christof Parr. Lecture 15: Elgamal encryption scheme.

Appendix

Genalyn and Kevin both believe that both parties contributed fairly to all parts of this coursework. Genalyn initially researched the algorithm and helped Kevin to understand it. Genalyn and Kevin then wrote the paper. Genalyn ran the worked examples, and explained them to Kevin. Kevin did the research for the comparison with RSA and talked about it with Genalyn. Kevin and Genalyn both worked on the software doing pair programming.