간략화한 Telegram의 상관 평문 문자에 대한 avalanche 검증

이지은[1] 강웅규[2] 정민[2] 최락용[1] 김광조[1]

[1]KAIST 전산학부 [2]대전과학고등학교

Avalanche test of related short messages in simplified Telegram

Jeeun Lee[1] Woong Gyu Kang[2] Min Jeong[2] Rakyong Choi[1] Kwangjo Kim[1]

[1]School of Computing, KAIST [2]Daejeon Science High School for the Gifted

요 약

Telegram은 사용자가 비밀 대화를 사용할 때 자체 설계한 MTProto의 단대단(end-to-end) 암호화를 통해 내용의 기밀성을 보장한다. 이 프로토콜은 오픈소스로 개발되어 암호화 과정을 직접 확인하고 안정성을 증명할 수 있다. 본 연구에서는 MTProto의 구조를 간략화하여 이와 유사하게 동작하는 구현물을 제작하였고, 이를 바탕으로 avalanche effect를 확인하여 preamble 부분이 없음에도 불구하고 안전함을 검증하였다. 국내에서 많이 사용되는 카카오톡의 경우 비밀 대화 기능을 구현하고 있으나 소스코드가 공개되지 않아 안정성의 검증이 불가능하다. 점차 오픈소스 커뮤니티를 통한 보안성 검증이 세계적 추세가 되고 있음을 고려하면, 국내 소프트웨어 업계에서도 이와 같은 방향성을 따르는 노력이 필요할 것으로 보인다.

I. 서론

2010년 이후 스마트폰 사용이 증가하면서 다양한 모바일 메신저 개발이 활발해졌다. 전 세계적으로 WhatsApp, Facebook Messenger, WeChat, LINE, Telegram 등이 강세를 보이고 있지만 국내에서는 카카오톡이 95% 이상의 점유율을 기록[1]하며 독점적인 위치를 차지하고 있다. 하지만 2014년 카카오톡 사찰 논란 이후 단대단(end-to-end) 비밀대화를 사용할 수 있는 Telegram의 사용자가 급증하였다.

Telegram은 사용자가 비밀 대화를 사용할 때 서버에 기록을 남기지 않고 암호화 메시지를 중계하는 역할만 한다. 따라서 메시지는 비밀키를 가지고 있는 단말기만 복호화할 수 있어 내용의 기밀성이 보장된다.

본 연구에서는 Telegram의 비밀 대화에 사용되는 MTProto의 구조를 간략화하여 이와 유사하게 동작하는 구현물을 제작하였다. 또한 이를 바탕으로 avalanche effect를 확인하여 preamble 부분이 없음에도 불구하고 안전함을 검증하였다. 마지막으로 향후 연구로써 MTProto의 보안성을 향상할 수 있는 개선방안에 대해 논의한다.

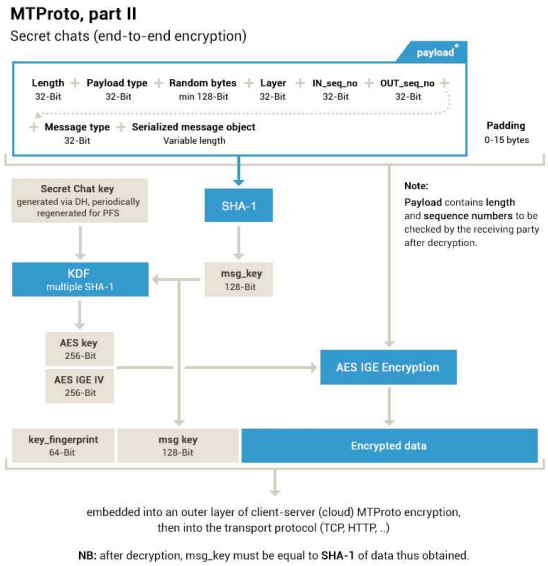
이 논문은 2016년도 정부(미래창조과학부)의 재원으로 정보통신기술진흥센터의 지원(B0101-16-1270, 생체모방 알고리즘(Bio-Inspired Algorithm)을 활용한 통신 기술 연구)과 2016년도 정부(미래창조과학부)의 재원으로 한국연구재단의 지원을 받아 수행된 연구임 (No. NRF-2015R1A2A2A01006812)

II. Telegram 비밀 대화 프로토콜

Telegram은 대다수의 모바일 메신저들과는 달리 오픈소스로 개발되어 암호화 과정을 직접 확인하고 안정성을 증명할 수 있다. 비밀 대화 통신을 위해 Telegram은 독자적으로 MTProto라는 프로토콜을 설계하였다. [그림 1][2]은 MTProto의 암호화 과정이다.

2.1 Alice가 Bob에게 비밀 대화 요청

Alice가 Bob과 암호 통신을 하기 위해 비밀 대화방 개설을 요청하고 싶다면, Alice는 `messages.getDhConfig()`를 실행하여 Diffie-Hell



[그림 1] MTProto의 암호화 과정

man 키 교환 방식에 사용할 소수 p와 적절한 생성원 g를 얻는다. 이 인자들을 저장해두고 매번 *Secret Chat key*를 생성하기 전에 `messages.getDhConfig()`를 실행하여, 만약 Alice가 최신상태라면 저장된 값을 그대로 쓴다. 만약 키가 100개 이상의 메시지를 암/복호화 했거나 일주일 이상 사용되었다면 이전 키를 버리고 새로운 키를 사용하여 완전한 전방향 보안성(Perfect Forward Security)을 강화한다.

Alice는 p가 2,048 비트 안전 소수(safe prime)인지, 즉, p와 $(p-1)/2$가 소수인지를 확인하고 g는 위수가 $(p-1)/2$인 순환군을 생성하는지, 즉, 이차잉여(quadratic residue) mod p인지 확인한다. 위의 과정을 거친 후 Alice는 2,048 비트 난수 a를 생성하고 $g_a = g^a \bmod p$를 계산한 후 `messages.requestEncryption()`을 실행한다. 만약 Alice가 난수 생성 시 완전한 난수 생성을 할 수 없다면, 생성하고 싶은 `random_length` 인자를 서버에게 넘겨 서버가 난수를 생성하게 해준다. 하지만 서버가 생성한 난수를 그대로 쓰는 것은 안전하지 않기 때문에 Alice가 생성한 불완전한 난수와 XOR하여 최종 난수를 다음과 같이 저장한다: `final_random = random XOR client_random`.

Bob이 `updateEncryption()`, `encryptedChatRequested()`, Alice의 기본 정보를 받으면 Alice의 비밀 대화 요청을 수락 또는 거절할지 정한다.

2.2 Bob이 Alice의 비밀 대화 요청 수락

Bob이 비밀 대화방 개설을 확인하면 Bob은 Alice로부터 Diffie-Hellman 키 교환에 사용될 인자들을 받는다. 그 후 Bob 또한 Alice의 방법처럼 2,048 비트 난수 b를 생성한다. 2.1의 마지막 과정에서 Alice로부터 g_a를 받았기 때문에 Bob이 b를 생성하면 $(g_a)^b \bmod p$를 계산하여 *Secret Chat key* 또한 생성할 수 있다. 만약 키가 256 바이트보다 작다면 0으로 패딩 하여 256 바이트로 만들고, `SHA1(key)`을 계산하여 마지막 64 비트만 fingerprint로 사용한다. 이 fingerprint는 소프트웨어 업데이트 과정에서 버그를 감지하기 위해 sanity-check로 사용된다.

Bob은 $g_b = g^b \bmod p$와 `key_fingerprint` 인자를 `messages.acceptEncryption()`에게 넘겨 실행한다. Alice가 `updateEncryption()`과 `encryptedChat()`을 받고, g_b로부터 *Secret Chat key*, $(g_b)^a \bmod p$를 생성한다. Bob과 마찬가지로 fingerprint를 계산하고 Bob으로부터 온 값과 동일한지 비교한다. 만약 동일하다면 메시지 수/발신을 시작하고, 값이 다르다면 `message.discardEncryption()`을 실행하고 사용자에게 알린다.

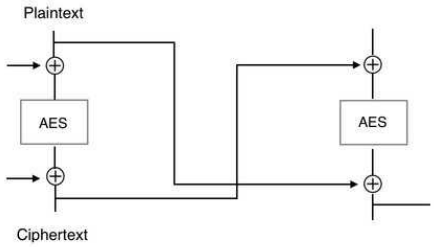
2.3 전송할 메시지 암호화

2.3.1 Key Derivation Function (KDF)

전송할 메시지들의 payload를 SHA-1 함수에 넣고 그 결과 하위 128 비트를 계산하여 `msg_key` 값으로 사용한다. 이로부터 평문 암호화에 사용할 키를 생성한다.

2.3.2 AES-256 Infinite Garble Extension (IGE)

IGE는 [그림 2][3]와 같이 동작하는 블록 암호 운영 모드 중 하나로, spoofing 공격, birthday 공격 등을 방지하기 위해 1978년 Campbell[4]에 의해 제안되었다. 운영 모드 진행 중 변화된(garbled) 데이터가 들어오는 경우, 이후 데이터 블



[그림 2] IGE mode for AES

록의 암호화 과정에서 제대로 된 값을 출력하지 못하는 infinite error propagation 성질을 가지고 있다. Telegram은 이 모드를 사용하는 AES-256 암호화를 위해 256 비트 키로는 aes_key, 256 비트 초기 벡터로는 aes_iv를 이용하였다. 최종적으로 key_fingerprint와 msg_key가 암호화 결과 앞에 붙으면서 암호문이 완성된다.

2.4 메시지 복호화 과정

복호화 과정은 앞의 과정들을 역순으로 수행한다. 암호화된 메시지를 수신할 때 받은 msg_key가 복호화된 메시지의 해시 값 하위 128 비트와 동일한지 반드시 확인하여야 한다.

III. 프로토콜 구현

3.1 구현 환경

운영체제: Windows

언어: C++11

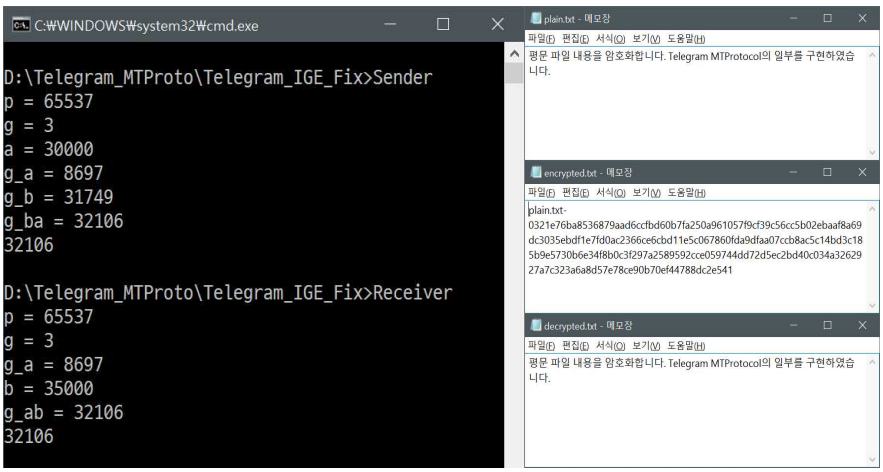
3.2 구현 내용

본 논문에서는 네트워크 통신을 생략하고 통신 내용을 파일에 기록하는 방식으로 암/복호화를 구현하였다. Diffie-Hellman 키 교환 방식에서는 소수 p를 $2^{16}+1$로 고정하고 생성원 g 값은 2, 3, 4, 5, 6, 7 중 선택할 수 있도록 하였다. p와 g 값을 확인할 때는 이차 상호 법칙(Quadratic Reciprocity Law)에 의해 $p \bmod 4g$ 조건만을 확인해 보면 된다.

실제 키 교환 과정은 BigInteger 라이브러리[5]를 이용하여 구현하였고, 이후 SHA-1 implementation in C++ 라이브러리[6]를 이용하여 KDF 함수를 통해 암호화에 쓸 키를 다음과 같이 생성하였다.

```
sha1_a = SHA1(msg_key + substr(key, x, 32));

sha1_b = SHA1(substr(key, 32+x, 16) + msg_key + substr(key, 48+x, 16));

sha1_c = SHA1(substr(key, 64+x, 32) + msg_key);

sha1_d = SHA1(msg_key + substr(key, 96+x ,32));

aes_key = substr(sha1_a, 0, 8) + substr(sha1_b, 8, 12) + substr(sha1_c, 4, 12);

aes_iv = substr(sha1_a, 8, 12) + substr(sha1_b, 0, 8) + substr(sha1_c, 16, 4) + substr(sha1_d, 0, 8);
```

AES-256 IGE 모드를 구현하기 위하여 Tiny AES128 in C 라이브러리[7]를 이용하였다. [그



[그림 3] Diffie-Hellman 키 교환 결과 및 평문 암/복호화

	평문	*Secret Chat key*	msg_key	암호문	암호문 1과의 해밍 거리
1	222222222222222	32106	e357c6a1205b3ead60f3 03d407a72851c318043f	45bf73fd58191e90 cc25c8dc9d5af144	-
2	222222222222223	32106	2d56451a2933fc94acc2 b469fab967b870ef4f74	71001bd1393b6367 c4b7b2c1db741f7b	66
3	ddddddddddddddd	32106	93875595643c025347b4 dab9345737b6fcdc0029	38fe07c7098e2358 f492aa55f3286def	65

[표 1] 상관성 있는 평문간 암/복호화 실험

림 3]의 스크린 샷에서는 키 교환 과정에 사용된 값을 확인할 수 있고 *encrypted.txt*와 *decrypted.txt*를 통해 암호화한 평문과 복호화 결과를 확인할 수 있다.

3.3 Avalanche 검증

Avalanche effect를 검증하기 위하여 상관성 있는 평문을 입력하고, 이에 따른 암/복호문을 분석하였다. [표 1]은 입력으로 사용한 3가지 평문과 그에 따른 키 및 암/복호문을 나타낸다. 첫 번째 평문은 '2' (= 0x32, 50)의 연속 값이고, 두 번째 평문은 첫 번째 평문에 1을 더한 값, 즉, 0x32⊕1, 51이다. 세 번째 평문은 'd' (= 0x64, 100)의 연속으로 첫 번째 경우의 2배 값에 해당한다.

이 경우 결과 암호문 사이에 어떤 관계가 나타나는지 알아보기 위해, 결과 암호문과 비교하고자 하는 암호문 1의 해밍 거리를 구해보았다. 그 결과 암호문간에는 어떠한 관계가 없음을 확인하였는데, 이는 동작 모드 상 preamble 부분이 없음에도 불구하고 SHA-1에 의해 msg_key가 달라지기 때문이다.

IV. 향후 연구 및 결론

비밀 대화를 위한 Telegram의 MTProto에서 msg_key는 payload의 해시 값에만 의존한다. 따라서 해시 함수에서 충돌이 발생한다면 메시지를 조작할 가능성이 있다. 프로토콜에서 사용되는 SHA-1은 현재 널리 사용되는 해시 함수 중 하나이나, 이론적으로는 충돌 가능성이 있음이 2005년에 발견[8]되었다. 따라서 안정성을 강화하기 위해 SHA-3로 대체하는 것이 향후 연구 계획이다.

Telegram은 오픈소스 소프트웨어로 비밀 대화를 위한 프로토콜의 설계와 구현이 모두 공개되어 있어, 사용자가 비밀 대화 프로토콜을 직접 구현하거나 구현된 프로토콜의 안정성을 확인할 수 있다. 반면 국내에서 많이 사용되는 카카오톡 등의 메신저는 비슷한 기능의 비밀 대화를 구현하고 있으나 소스코드가 공개되지 않아 안정성의 검증이 불가능하다. 점차 오픈소스 커뮤니티를 통한 보안성 검증이 세계적 추세가 되고 있음을 고려하면, 국내 소프트웨어 업계에서도 이와 같은 방향성을 따르는 노력이 필요할 것으로 보인다.

[참고문헌]

[1] WiseApp. http://www.wiseapp.co.kr

[2] Telegram. https://core.telegram.org/mtproto

[3] AES IGE. https://www.mgp25.com/AESIGE

[4] Campbell, C.: Design and specification of cryptographic capabilities. IEEE Communications Society Magazine. 16, 15 - 19 (1978).

[5] BigInteger 라이브러리. https://github.com/panks/BigInteger

[6] SHA-1 implementation in C++ 라이브러리. https://bitbucket.org/voxelstorm/sha1

[7] Tiny AES128 in C 라이브러리. https://github.com/kokke/tiny-AES128-C

[8] Wang, X., Yin, Y.L., Yu, H.: Finding Collisions in the Full SHA-1. Presented at the Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings (2005).