暗号に基づくワンタイムプログラムの実現

代表研究者　西出 隆志　筑波大学　システム情報系　准教授

1　はじめに

Goldwasser ら [GKR08] は One-Time Memroy (OTM) と呼ばれるハードウェアデバイスの存在を仮定し Garbled Circuit と組み合わせることで OTP を構成できることを示した。しかしデバイスを OTP 実行者に物理的に配布しなければならないという制限がある。北村ら [KSNO17] はデバイスの使用を避けるためクラウドサーバらに OTM 内のデータを秘密分散し，データがアクセスされた後に消すことで OTM の機能をクラウドサーバに分散させることを提案した。ここでは [西 19] の内容に基づき [KSNO17] の手法をさらに発展させた方式について報告する。

OTP 関連研究. Gunupudi ら [GT08] は TPM を拡張することで非対話 Oblivious Transfer を実装し，それにより非対話 Garbled Circuit (GC) を実現した。その後 Goldwasser ら [GKR08] は Oblivious Transfer とほぼ同様に動作する One-Time Memory と呼ばれる耐タンパデバイスの存在を仮定し，GC を少し変形して組み合わせることで OTP を実現した。北村ら [KSNO17] は耐タンパデバイスではなくクラウドサーバに OTM の機能を分散させることで [GKR08] と同様に GC を用いて OTP を構成した。Goyal ら [GG17] は Proof-of-Stake に基づく Blockchain と Witness Encryption [GGSW13] を仮定し，これらから OTM を構成し [GKR08] と同様 GC を用いて OTP を構成し

た[1]。Zhao ら [ZCD+19] は Trusted Execution Environment (TEE)[2] の一つ Intex-TXT[Gre12] と TPM[tpm] を用いて OTM を実装し，そして GC と組み合わせることで OTP を実装した。また OTM の実行だけでなくプログラムの実行全体を TEE により保護する方法での OTP 実装も提案している。

2 準備

2.1 Garbled Circuit

Garbled Circuit (GC) は入力を秘匿したままでの計算を可能とする手法 (garbling scheme) である [Yao86]。今，回路族 $\{\mathcal{C}_n\}_{n\in\mathbb{N}}$ の $\mathcal{C}_n$ の各回路は n bit の入力を受け取る回路とすると garbling scheme GC は以下のアルゴリズムからなる。

Definition 1 (Garbling Scheme GC).

$\mathsf{GC.Garble}(1^\lambda,\ C \in \mathcal{C}_n)$: セキュリティパラメータ λ と回路 C を受け取り，*garbled circuit* C_{gc} とラベルと呼ばれる[3] *garbled input* $\{\ell_i^{(0)},\ \ell_i^{(1)}\}_{1\le i\le n}$ を出力する。

$\mathsf{GC.Eval}(C_{\mathrm{gc}},\ \{\ell_i^{(x_i)}\}_{1\le i\le n})$: *garbled circuit* C_{gc} と入力ビット列 $x = (x_1,\ldots,x_n) \in \{0,1\}^n$ に相当するラベルを受け取り $C(x)$ を出力する。

$\mathsf{GC.Garble}$ では回路 C はビット列を入力とする論理 *(Boolean)* 回路として表現されていることを仮定している。

[1] この方式では OTP 実行者は OTP への入力 (あるいは入力への commitment) を Blockchain に書き込む。その後ある程度伸びた条件を満たす Blockchain を Witness として利用し，GC を実行するために必要な (ラベルと呼ばれる) 情報を含む Witness Encryption の暗号文を復号する。ハードウェアに依存しない方式であるが複雑な条件を判定する Witness Encryption を必要としており，現時点では実用レベルにはまだ近くないと考えられる。

[2] これは特別なデバイスではなく CPU が提供する機能であり，ハードウェアの支援によりプログラムの安全な実行環境を実現する。

[3] token また key と呼ばれることもある [GIS+10, BHR12]。

Functionality OTM

- On input $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (\ell^{(0)}, \ell^{(1)}))$ from party P_i, send $(P_i, P_j, \mathsf{sid}, \mathsf{id})$ to P_j and store the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (\ell^{(0)}, \ell^{(1)}))$.
- On receiving $(P_i, \mathsf{sid}, \mathsf{id}, b(\in \{0,1\}))$ from party P_j, if a tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (\ell^{(0)}, \ell^{(1)}))$ exists, return $(P_i, \mathsf{sid}, \mathsf{id}, \ell^{(b)})$ to P_j and delete the tuple $(P_i, P_j, \mathsf{sid}, \mathsf{id}, (\ell^{(0)}, \ell^{(1)}))$. Else, do nothing.

Fig. 1. Ideal Functionality for OTM [GIS+10]

本提案では OTP に適した適応的安全性を満たす GC [BHR12] を使うことを想定する。

2.2 Goldwasser らによる OTP 方式 [GKR08]

ここでは Goldwasser による GC と OTM を用いた OTP 方式 GKR を説明する。[GKR08] では OTM と呼ばれる Fig. 1 の機能を提供するハードウェアデバイスを仮定する。

つまり OTM は内部に $(\ell^{(0)}, \ell^{(1)})$ の 2 つのラベルを持ち，指定された b に対するラベル $\ell^{(b)}$ を一度だけ返す。この GC と OTM を用いて [GKR08] では以下のように回路 $C \in \mathcal{C}_n$ に対する OTP を構成する。

1. OTP 作成者 $\mathsf{G}_{\mathrm{otp}}$ は $\mathsf{GC.Gable}(1^\lambda, C)$ を実行し，C_{gc} とラベル $\{\ell_i^{(0)}, \ell_i^{(1)}\}_{1\le i\le n}$ を得る。
2. $\mathsf{G}_{\mathrm{otp}}$ はそれぞれのラベルの組 $\{\ell_i^{(0)}, \ell_i^{(1)}\}$ を OTM_i に埋め込み，$\langle C_{\mathrm{gc}}, \{\mathsf{OTM}_i\}_{1\le i\le n}\rangle$ を C の OTP として OTP 実行者 $\mathsf{E}_{\mathrm{otp}}$ に送る。
3. $\mathsf{E}_{\mathrm{otp}}$ は入力ビット列 $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ に対応するラベル $\{\ell_i^{(x_i)}\}_{1\le i\le n}$ を各 OTM_i から取得し，$\mathsf{GC.Eval}(C_{\mathrm{gc}}, \{\ell_i^{(x_i)}\}_{1\le i\le n})$ を実行し $C(x)$ を得る。

2.3 Message Authentication Code (MAC)

MAC は次のアルゴリズムの組から構成される。

Definition 2 (Message Authentication Code (MAC)).

$\mathsf{MAC.KeyGen}(1^\lambda)$: セキュリティパラメータ λ を受け取り鍵 k を出力する。ここで k のビット長 $|k| \geq n$ となっている。

$\mathsf{MAC.Mac}(k,\ m)$: 鍵 k とメッセージ $m \in \{0,1\}^*$ を受け取り，m の $MAC\ tag$ $\mathsf{tag}(m)$ を出力する。

$\mathsf{MAC.Vrfy}(k,\ m,\ \mathsf{tag})$: 鍵 k とメッセージ m と $MAC\ tag$ tag を受け取り，tag が m の正しい $MAC\ tag$ なら 1 を，そうでなければ 0 を出力する。

3 提案方式

3.1 クラウドに基づくワンタイムプログラム (OTP)

[KSNO17] は [GKR08] の OTM の機能を複数のクラウドサーバに分散することにより OTP を構成した [4]。ここでは [KSNO17] の OTP 方式に対し，以下の改良を加える。

- クラウドサーバに保存するラベルのシェアに対して，MAC を付加することにより安全性を高める。
- クラウドサーバ/ユーザ間の Oblivious Transfer を準同型暗号ベースにし，各クラウドサーバとの個別の通信ではなく単に同一のデータをユーザはクラウドサーバらへブロードキャストするだけで各入力ビットに対するラベルを得られるようにする。

[KSNO17] では各ラベルのペア $(\ell_i^{(0)},\ \ell_i^{(1)})$ を Shamir の秘密分散 [Sha79] により分割して各クラウドサーバに渡している。それに基づき回路 $C \in \mathcal{C}_n$ に対する OTP 作成処理 (Fig. 2) を提案する。

また $\mathsf{E}_{\mathrm{otp}}$ による OTP 実行処理を Fig. 3 に記述する。

[4] ここではクラウドサーバが OTM に相当する分散サービスを提供することで OTP 作成者からサービス料金を得るようなモデルを想定している。

1) $\mathsf{G}_{\mathrm{otp}}$ は garbling scheme により $C_{\mathrm{gc}}, \{\ell_i^{(0)},\ \ell_i^{(1)}\}_{1\le i\le n}$ を得る。またランダムな鍵 sk_C を生成する。
2) $\mathsf{G}_{\mathrm{otp}}$ は (k,m)-秘密分散により，$\{\ell_i^{(0)},\ \ell_i^{(1)}\}_{1\le i\le n}$ を $\{\ell_{i,j}^{(0)},\ \ell_{i,j}^{(1)}\}_{1\le i\le n,1\le j\le m}$ のシェアに分割する。
3) $\mathsf{G}_{\mathrm{otp}}$ は C の各入力ワイヤに ID として $w_i = H(sk_C \parallel i)$ を割り当て (H は適切なハッシュ関数)，各ラベルのシェア $\ell_{i,j}^{(b)}$ の MAC tag $\mathsf{tag}(\ell_{i,j}^{(b)})$ を次のように計算する。ここで $\parallel$ はビット列の結合を意味する。

$$\mathsf{tag}(\ell_{i,j}^{(b)}) \leftarrow \mathsf{MAC.Mac}(sk_C,\ \ell_{i,j}^{(b)} \parallel w_i \parallel b)$$

4) クラウドサーバ群を $\{\mathsf{S}_j\}_{1\le j\le m}$ とする。このとき $\mathsf{G}_{\mathrm{otp}}$ は S_j に $\{w_i,\ \ell_{i,j}^{(0)},\ \ell_{i,j}^{(1)},\ \mathsf{tag}(\ell_{i,j}^{(0)}),\ \mathsf{tag}(\ell_{i,j}^{(1)})\}_{1\le i\le n}$ を送る。
5) $\mathsf{G}_{\mathrm{otp}}$ は C に対する OTP として $\langle n,\ sk_C,\ C_{\mathrm{gc}},\ \{\mathsf{S}_j\}_{1\le j\le m}\rangle$ を OTP 実行者 $\mathsf{E}_{\mathrm{otp}}$ に渡す。

Fig. 2. OTP 作成処理

3.2 秘密分散 [Sha79] の (k,m) の設定

[KSNO17] で既に言及されているように適切な (k,m) の設定がなされない場合は問題が起こりうる。

今あるラベル $(\ell^{(0)},\ell^{(1)})$ の秘密分散に $(6,9)$-秘密分散が使用されているとする。このとき敵が 6 台のクラウドサーバをコラプトしない限り両方のラベルが敵に得られることはないと通常なら期待する。しかし実際には次に示すように 3 台のクラウドサーバをコラプトするだけで両方のラベル $(\ell^{(0)},\ \ell^{(1)})$ を敵は得ることができてしまう。

1. Table 1 のように $(\ell^{(0)},\ \ell^{(1)})$ が $\{\mathsf{S}_i\}_{1\le i\le 9}$ によって $(6,9)$-秘密分散されているとする。

[5] OTP に与える引数 x_i をクラウドサーバから秘匿する必要が無い場合は Oblivious Transfer は不要となる。このときは単純に x_i に対応するラベルのシェアを要求することでより効率的に実行できる。

[6] ここで S_j 内では適切な排他制御により，あるスレッドが w_i のデータ組に対する処理を行っている間，他のスレッドが $\{w_i,\ \ell_{i,j}^{(0)},\ \ell_{i,j}^{(1)},\ \mathsf{MAC}(\ell_{i,j}^{(0)}),\ \mathsf{MAC}(\ell_{i,j}^{(1)})\}$ へアクセスすることが禁止される実装となっていることを仮定する。

1) $\mathsf{E}_{\mathrm{otp}}$ は加法準同型暗号の使い捨て鍵 $(pk,\ sk)$ を生成する。$\mathsf{E}_{\mathrm{otp}}$ は $x = (x_1, \ldots, x_n) \in \{0,1\}^n$ を入力として OTP を実行するとする。
2) x_i に対するラベルのシェア $\ell_{i,j}^{(x_i)}$ を得るために $\mathsf{S}_j (1 \le j \le m)$ に対して x_i の pk による暗号文 $E_{pk}(x_i)$ と $x_i \in \{0,1\}$ であることを示すゼロ知識証明 π_{x_i} を作成し，$\langle w_i,\ pk,\ E_{pk}(x_i),\ \pi_{x_i} \rangle$ を $\mathsf{S}_j (1 \le j \le m)$ に対してブロードキャストする。
3) S_j は w_i に対応するデータが無いあるいは π_{x_i} が無効な場合は無視する。それ以外の場合は次の Oblivious Transfer に相当する公開鍵 pk での準同型演算を行う [5]。

$$c_{i,j}^{(0)} \leftarrow E_{pk}(1-x_i) \cdot \ell_{i,j}^{(0)} + E_{pk}(x_i) \cdot r_{i,j}^{(0)}, \qquad m_{i,j}^{(0)} \leftarrow E_{pk}(1-x_i) \cdot \mathsf{tag}(\ell_{i,j}^{(0)}) + E_{pk}(x_i) \cdot \gamma_{i,j}^{(0)}$$

$$c_{i,j}^{(1)} \leftarrow E_{pk}(x_i) \cdot \ell_{i,j}^{(1)} + E_{pk}(1-x_i) \cdot r_{i,j}^{(1)}, \qquad m_{i,j}^{(1)} \leftarrow E_{pk}(x_i) \cdot \mathsf{tag}(\ell_{i,j}^{(1)}) + E_{pk}(1-x_i) \cdot \gamma_{i,j}^{(1)}$$

ここで $r_{i,j}^{(0)},\ \gamma_{i,j}^{(0)},\ r_{i,j}^{(1)},\ \gamma_{i,j}^{(1)}$ は乱数である。そして S_j は $\{w_i,\ \ell_{i,j}^{(0)},\ \ell_{i,j}^{(1)},\ \mathsf{tag}(\ell_{i,j}^{(0)}),\ \mathsf{tag}(\ell_{i,j}^{(1)})\}$ を削除し [6]，$\langle c_{i,j}^{(0)},\ m_{i,j}^{(0)},\ c_{i,j}^{(1)},\ m_{i,j}^{(1)} \rangle$ を $\mathsf{E}_{\mathrm{otp}}$ へ返信する。
4) $\mathsf{E}_{\mathrm{otp}}$ は $c_{i,j}^{(x_i)}, m_{i,j}^{(x_i)}$ を sk で復号し，$\mathsf{tag}(\ell_{i,j}^{(x_i)})$ が正しいか $\mathsf{MAC.Vrfy}$ と sk_C を使って検証する。
5) MAC tag の検証が正しい場合は，シェアの集合 $\left\{\ell_{i,j}^{(x_i)}\right\}_j$ から $\ell_i^{(x_i)}$ を復元し，$\mathsf{GC.Eval}(C_{\mathrm{gc}},\ \{\ell_i^{(x_i)}\}_{1 \le i \le n})$ を実行し $C(x)$ を得る。

Fig. 3. OTP 実行処理

2. 敵が $\mathsf{S}_1, \mathsf{S}_2, \mathsf{S}_3$ をコラプトした場合，敵は $\{\ell_i^{(b)}\}_{1 \le i \le 3, b=0,1}$ のシェア (灰色部分) を得る。

3. 敵は $\mathsf{S}_4, \mathsf{S}_5, \mathsf{S}_6$ に対して 1 のラベルのシェアを要求し $\{\ell_i^{(1)}\}_{4 \le i \le 6}$ のシェアを得る (灰色部分)。

4. 敵は $\mathsf{S}_7, \mathsf{S}_8, \mathsf{S}_9$ に対して 0 のラベルのシェアを要求し $\{\ell_i^{(0)}\}_{7 \le i \le 9}$ のシェアを得る (灰色部分)。

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8	S_9
$\ell^{(0)}$	$\ell_1^{(0)}$	$\ell_2^{(0)}$	$\ell_3^{(0)}$	$\ell_4^{(0)}$	$\ell_5^{(0)}$	$\ell_6^{(0)}$	$\ell_7^{(0)}$	$\ell_8^{(0)}$	$\ell_9^{(0)}$
$\ell^{(1)}$	$\ell_1^{(1)}$	$\ell_2^{(1)}$	$\ell_3^{(1)}$	$\ell_4^{(1)}$	$\ell_5^{(1)}$	$\ell_6^{(1)}$	$\ell_7^{(1)}$	$\ell_8^{(1)}$	$\ell_9^{(1)}$

Table 1. ラベル $\ell^{(0)}, \ell^{(1)}$ の $(6, 9)$-秘密分散

このようにクラウドサーバ3台をコラプト後，残りのクラウドサーバに $0, 1$ 異なる両方の (不正な) 要求を送ることによって敵は $(\ell^{(0)}, \ell^{(1)})$ のシェアをそれぞれ6個集めることができてしまう [7)]。

このように敵に両方のラベルを得られ，OTP を2回以上実行されてしまうことを防ぐための適切な (k, m) の設定を求めておく。ここで敵は多くて t 台のクラウドサーバをコラプト可能と仮定する。このとき (k, m) は以下の条件を満たす必要がある。

1. $t < k \leq m$
2. $m + t < 2k$ (t 台コラプト後に上記の攻撃により両方のラベルを得られなくするための条件)

よって $k = t + \alpha$ $(\alpha \geq 1)$ とすると

$$m < 2k - t = 2t + 2\alpha - t = t + 2\alpha$$

と設定すればよいことになる [8)]。

参考文献

[BHR12] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *Asiacrypt*, pages 134–153. Springer, 2012.

[GG17] Rishab Goyal and Vipul Goyal. Overcoming cryptographic impossibility results using blockchains. In *TCC*, pages 529–561. Springer, 2017.

[7)] クラウドサーバ間で連携するプロトコルをさらに組み込むことによってこのような攻撃は防ぐことができるかもしれない。しかし実世界への配置を容易とするためにクラウドサーバ間での通信を不要な設計としている。

[8)] よって冗長性を自由に大きくすることはできないというトレードオフが存在する。

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *STOC*, pages 467–476. ACM, 2013.

[GIS+10] Vipul Goyal, Yuval Ishai, Amit Sahai, Ramarathnam Venkatesan, and Akshay Wadia. Founding cryptography on tamper-proof hardware tokens. In *TCC*, pages 308–326. Springer, 2010.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N Rothblum. One-time programs. In *CRYPTO*, pages 39–56. Springer, 2008.

[Gre12] James Greene. Intel trusted execution technology. *Intel Technology White Paper*, 2012.

[GT08] Vandana Gunupudi and Stephen R Tate. Generalized non-interactive oblivious transfer using count-limited objects with applications to secure mobile agents. In *Financial Cryptography and Data Security*, pages 98–112. Springer, 2008.

[KSNO17] Takuya Kitamura, Kazumasa Shinagawa, Takashi Nishide, and Eiji Okamoto. One-time programs with cloud storage and its application to electronic money. In *ASIA Public-Key Cryptography (APKC)*, pages 25–30. ACM, 2017.

[Sha79] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[tpm] Trusted Computing Group (TCG). TPM main specification. Main specification, Trusted Computing Group, May 2009. `http://www.trustedcomputinggroup.org`.

[Yao86] Andrew Yao. How to generate and exchange secrets. In *FOCS*, pages 162–167. IEEE, 1986.

[ZCD+19] Lianying Zhao, Joseph I Choi, Didem Demirag, Kevin RB Butler, Mohammad Mannan, Erman Ayday, and Jeremy Clark. One-time programs made practical. In *Financial Cryptography and Data Security*, pages xx–xxx. Springer, 2019.

[西 19] 西出隆志. ワンタイムプログラムとそのシュノア署名への応用. In *Technical report of IEICE*. ISEC 研究会, 3 月, 2019.

＜発表資料＞

題名	掲載誌・学会名等	発表年月
Outsourced Private Function Evaluation with Privacy Policy Enforcement	17th IEEE International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)	2018 年 8 月
ワンタイムプログラムとそのシュノア署名への応用	Technical report of IEICE, ISEC	2019 年 3 月
ノンスベースのハイブリッド暗号	Technical report of IEICE, ISEC	2019 年 3 月