完全準同型暗号を用いた秘匿LTLオンラインモニタリング

伴野 良太郎[1,a)]　松岡 航太郎[1]　松本 直樹[2]　Song Bian[1]　和賀 正樹[1]　末永 幸平[1]

概要：本研究では，Alice が Bob にデータを提供し，Bob が Alice のデータに対して線形時相論理（LTL）によって記述された仕様を用いてオンラインモニタリングを行う際に，Alice のデータやモニタリング結果を Bob から秘匿し，また Bob の仕様を Alice から秘匿するプロトコル（秘匿 LTL オンラインモニタリング）を提案する．我々の知る限りにおいて，本手法は秘匿 LTL オンラインモニタリングを行う初めてのプロトコルである．本研究では，LTL 式を変換し得られた DFA を，完全準同型暗号の一種である TFHE を用いて実行する．その手法として，(i) 入力を末尾から用いる既存のオフラインアルゴリズムを DFA を反転させることでオンラインの設定に転用する手法，及び (ii) DFA の現状態を暗号文として保持し，各入力によって起こる状態の遷移を TFHE で可能な操作を用いて暗号文上で行う手法の 2 つを提案した．またケーススタディとして，血糖値のモニタリングを行う LTL 式と，1 型糖尿病患者のシミュレーションデータを用い，値の計測間隔である 1 分よりも速く各値の処理を行えることを確認した．

キーワード：モニタリング, 準同型暗号, 線形時相論理, オートマトン, 秘匿計算

Oblivious Online Monitoring for Linear Temporal Logic Specification through Fully Homomorphic Encryption

Ryotaro Banno[1,a)]　Kotaro Matsuoka[1]　Naoki Matsumoto[2]　Song Bian[1]　Masaki Waga[1]
Kohei Suenaga[1]

Abstract: We propose a protocol to achieve *oblivious Linear Temporal Logic (LTL) online monitoring*. Under an oblivious LTL monitoring setting, Alice holds private data, and Bob monitors her data with private specification described by LTL. With an oblivious LTL protocol, we can prevent Bob from learning the data and the monitoring results from Alice, while also guaranteeing that Alice knows nothing about the specification of Bob. To the best of our knowledge, this is the first privacy-preserving protocol for LTL online monitoring. We first convert the LTL formula to a DFA. Then, the DFA is evaluated over TFHE, one kind of fully homomorphic encryption schemes. We propose two implementations: i) we tweak the existing offline algorithm to read the inputs from back to front by reversing the DFA; and ii) we keep the current state of the DFA as ciphertexts, and perform state transitions with homomorphic operations.

Keywords: monitoring, homomorphic encryption, linear temporal logic, automaton, secure computation

1. 序論

Internet of Things（IoT）の普及によって，IoT デバイスから送られるデータを監視し，異常がある場合には警告を出すようなモニタリングシステムの需要が増大している．特にウェアラブルデバイスを通じて血糖値や心電図の値を計測しモニタリングや解析を行うようなシステムは Medical Internet of Things（MIoT）とも呼ばれ，継続的な健康管理によって人々の健康増進につながると期待されている [6]．このようなシステムにおいて監視仕様を記述する手法として時相論理がある [3], [14]．特に線形時相論

[1] 京都大学大学院 通信情報システム専攻
Department of Communications and Computer Engineering, Kyoto University
[2] 京都大学大学院 知能情報学専攻
Department of Intelligence Science and Technology, Kyoto University
[a)] banno@fos.kuis.kyoto-u.ac.jp

理（Linear Temporal Logic, LTL）と呼ばれる論理のうち，ある種の性質を満たす論理式では，対応するモニタを有限状態オートマトン（DFA）として構成できる．これを用いることで，デバイスからのデータが想定した仕様を満たしているかを判別できる．

MIoT においては，個人の生体情報という極めてセンシティブなデータを扱うため，データのセキュリティやプライバシーが問題となる．例えばセンサから得られたデータをクラウドサーバに送信して処理を行う場合，通信路やクラウドサーバへの攻撃によって，これらのデータが漏洩する恐れがある．一方でエッジデバイス上でモニタリングを行う場合，ユーザのデータは秘匿されるものの，デバイス上で動作するモニタリング手法がユーザに漏洩する恐れがある．監視手法が高度なものである場合や，非公開のデータを学習して得られたパラメタを使用している場合などには，監視手法の漏洩は経済的な損失を含む問題となりうる．以下では，ユーザのデータをサーバから秘匿し，かつサーバ上で動作するモニタをユーザから秘匿して行うようなモニタリングを**秘匿 LTL モニタリング**と呼ぶ．

既存研究では，TFHE（Torus Fully Homomorphic Encryption）と呼ばれる完全準同型暗号を用いることで，DFA の評価を行う手法が知られている [4]．しかしこの手法では DFA への入力を末尾から用いるため，DFA への入力データが DFA の実行開始時点で全て得られている必要があった（オフライン）．一方で MIoT では体調の急変などを速やかに検出し適切な処置を講じる必要があるため，計測値をリアルタイムに処理することが必要である（オンライン）．同じ秘匿の条件を満たした上でオンラインに DFA を実行する手法は，我々の知る限り知られていない．

本研究では，TFHE を用いて秘匿オンライン LTL モニタリングを可能にするアルゴリズムを 2 種類提案する．一方は既存のオフラインアルゴリズムをオンラインアルゴリズムに転用する手法で，予め DFA を反転させてからアルゴリズムを適用することで，実効的に入力を先頭から読み込むことを可能にする．他方は TFHE 上で許される暗号文への操作を用いて平文時の DFA のオンライン実行と等価な処理を行うことで，暗号文上での DFA のオンライン実行を可能にする．提案手法の実用性を確認するために，血糖値のモニタリングを行う LTL 式と，1 型糖尿病患者の血糖値のシミュレーションデータを用いて実験を行った．その結果いずれの手法でも，血糖値の計測間隔である 1 分よりも短い時間で，各計測値の処理を行えることが分かった．

本論文の構成は次のとおりである．まず 2 節で準同型暗号や時相論理について説明し，議論の準備を行う．続いて 3 節で本研究において仮定するプロトコル・セキュリティモデルについて説明する．4 節では提案するオンラインアルゴリズムの 1 つである反転アルゴリズムについて説明し，5 節ではもう 1 つのアルゴリズムである Forward LUT アルゴリズムについて述べる．6 節では提案手法の実験結果を示し，最後に 7 節において関連研究を述べる．

2. 準備

2.1 準同型暗号

準同型暗号は，暗号文上での計算を復号せずに行うことができる暗号である．特に**完全準同型暗号（FHE）**は，暗号文に対して任意の演算を実現可能であるものを指し，Gentry [8] によって初めて構成された．FHE において暗号文を生成する際には，ノイズと呼ばれる乱数を加えることによってその安全性を保証する．暗号文のノイズは，FHE における演算を経る毎に大きくなり，最終的には復号が不可能になってしまう．これを防ぐために，FHE では **bootstrapping** と呼ばれる操作を導入する．bootstrapping では準同型暗号の上で復号操作を行うことにより，このノイズを定数に落とす．

本研究では TFHE [4] と呼ばれる FHE を使用する．TFHE はバイナリ演算に特化した FHE の 1 つで，bootstrapping を他の FHE と比べ高速（10 ミリ秒オーダー）に実行できるという特長を持つ．

以下では $\mathbb{B} = \{0, 1\}$ をブール値の集合とし，正整数 N について $\mathbb{B}_N[X]$ を最高で $N-1$ 次の $\mathbb{B}$ を係数とする多項式とする．

N は TFHE のセキュリティパラメタとする．TFHE には次のような暗号文の種類が存在する．

TLWE　平文 $m \in \mathbb{B}$ を表す暗号文である．後述する BOOTSTRAPPING の入力に用いられる．

TRLWE　平文 $m[X] \in \mathbb{B}_N[X]$ を表す暗号文である．

TRGSW　平文 $m \in \mathbb{B}$ を表す暗号文である．後述する CMUX の選択制御入力に用いられる．

秘密鍵 SK により平文 m を暗号化した暗号文を $\mathrm{Enc}_{\mathrm{SK}}(m)$，暗号文 c を復号した平文を $\mathrm{Dec}_{\mathrm{SK}}(c)$ により表す．なお暗号文の種類は文脈に依存して判断する．また $\textsc{Trivial}_i$ を，定数項のみが $i \in \{0, 1\}$ であり，残りの係数が全て 0 であるような多項式を表す TRLWE のサンプルとする．$\textsc{Trivial}_i$ は秘密鍵 SK にも乱数生成器にも依存せず生成される．

TFHE には次のような暗号文上の操作が存在する．

加法　TLWE・TRLWE・TRGSW のいずれについても加法準同型性が成り立つ．特に TRLWE について，平文 $m_1[X] = \sum_{i=0}^{N-1} m_{1i} X^i$, $m_2[X] = \sum_{i=0}^{N-1} m_{2i} X^i$ を表すような暗号文 c_1, c_2 について $c = c_1 + c_2$ は $m = \sum_{i=0}^{N-1} (m_{1i} \oplus m_{2i}) X^i$ を表す暗号文となる．ここで $\oplus$ は排他的論理和を表す．

CMux(c, d_1, d_0)　TRGSW の暗号文 c と TRLWE の暗号文 d_1, d_0 を受け取り，$\mathrm{Dec}_{\mathrm{SK}}(c) = 1$ のときに $\mathrm{Dec}_{\mathrm{SK}}(d_1)$ を表す TRLWE の暗号文を出力し，$\mathrm{Dec}_{\mathrm{SK}}(c) = 0$ のときに $\mathrm{Dec}_{\mathrm{SK}}(d_0)$ を表す TRLWE の暗号文を出力する．

CyclicShift$_k(c)$　TRLWE の暗号文 c を受け取り，それが

表す平文 $m[X] = \sum_{i=0}^{N-1} m_i X^i$ について，次のような平文 $m'[X] = \sum_{i=0}^{N-1} m'_i X^i$ を表す暗号文を生成する．

$$m'_i = \begin{cases} m_{i+k} & (i+k < N) \\ m_{i+k-N} & (i+k \geq N) \end{cases}$$

Lookup$_{d_0,d_1,\ldots,d_{2^n-1}}(c_0, c_1, \ldots, c_{n-1})$　2^n 個の TRLWE の暗号文 $d_0, d_1, \ldots, d_{2^n-1}$ と，n 個の TRGSW の暗号文 $c_0, c_1, \ldots, c_{n-1}$ を受け取り，$d_0, \ldots, d_{2^n-1}$ を n 段の LUT を構成するエントリとみなし，$c_0, \ldots, c_{n-1}$ を LUT への入力とみなして，選ばれた LUT エントリの値を表す TRLWE の暗号文 d を出力する．この LUT は $2^n - 1$ 個の CMux を n 段に構成することによって実現される．

SampleExtract$_k(c)$　整数値 $k < N$ と TRLWE の暗号文 c を受け取り，TLWE の暗号文 d を出力する．ここで $\mathrm{Dec}_{\mathrm{SK}}(d)$ は $\mathrm{Dec}_{\mathrm{SK}}(c)$ の X^k の係数と一致する．

Bootstrapping$_{\mathbf{BK}}(c)$　入力として bootstrapping 鍵 BK と TLWE の暗号文 c を受けとり，出力として TRLWE の暗号文 d を与える．$\mathrm{Dec}_{\mathrm{SK}}(d)$ の定数項は $\mathrm{Dec}_{\mathrm{SK}}(c)$ と一致する．出力の暗号文が含むノイズは，TFHE のパラメタから定まる定数となる．この操作は加法・CMux・CyclicShift・SampleExtract 等の操作と比べ低速である．

TLWE-to-TRLWE$_{\mathbf{KSK}}(c)$　key switching 鍵 KSK と TLWE の暗号文 c を受け取り，出力として TRLWE の暗号文 d を与える．このとき $\mathrm{Dec}_{\mathrm{SK}}(d)$ の定数項は $\mathrm{Dec}_{\mathrm{SK}}(c)$ と一致し，残りの係数は 0 である．

2.2 時相論理

時相論理は, 時間を経るに従い発生する事象間の関係について言及できる論理である [5]．時相論理を用いることで，時系列データに対する仕様を記述し，その検証を行うことができる．本研究では**線形時相論理（Linear Temporal Logic, LTL）**を扱う．

本研究で用いる LTL 式は次のように定義される [5], [7].

$$\phi, \psi ::= \top \mid \bot \mid p \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi \mid \mathsf{X}\phi \mid \mathsf{F}\phi \mid \mathsf{G}\phi \mid \phi \mathsf{U} \psi \mid \phi \mathsf{R} \psi$$

ここで ϕ, ψ は LTL 式上を動くメタ変数であり，p は命題変数の集合 AP 上を動くメタ変数である．$\top$, $\bot$, $\neg$, $\wedge$, $\vee$ は命題論理と同様の意味を持つ．$\mathsf{X}, \mathsf{F}, \mathsf{G}, \mathsf{U}, \mathsf{R}$ は LTL に特有の時相演算子で，各々「次の時点で」(X)・「将来のどこかで」(F)・「将来に渡りずっと」(G)・「ある時点までずっと」(U, R) を表す．また略記として以下を導入する．

- $\mathsf{G}_{[n,m]}\phi \overset{\mathrm{def}}{\iff} \overbrace{\mathsf{X}\ldots\mathsf{X}}^{n \text{ 個の } \mathsf{X}}(\phi \wedge \overbrace{\mathsf{X}(\phi \wedge \mathsf{X}(\cdots \wedge \mathsf{X}\phi))}^{m-n \text{ 個の } \mathsf{X}})$
- $\mathsf{F}_{[n,m]}\phi \overset{\mathrm{def}}{\iff} \overbrace{\mathsf{X}\ldots\mathsf{X}}^{n \text{ 個の } \mathsf{X}}(\phi \vee \overbrace{\mathsf{X}(\phi \vee \mathsf{X}(\cdots \vee \mathsf{X}\phi))}^{m-n \text{ 個の } \mathsf{X}})$

任意の $w \in (2^{\mathrm{AP}})^\omega$ について，LTL 式 ϕ が次を満たす時，ϕ は **safety** である（**safety LTL**）と呼ぶ [9]：$w \not\models \phi$ ならば，ある w の接頭辞 $w' \in (2^{\mathrm{AP}})^*$ が存在し，任意の $w'' \in (2^{\mathrm{AP}})^\omega$ について $w'w'' \not\models \phi$ を満たす．ただしここで $w \in (2^{\mathrm{AP}})^\omega$ は無限列 $w : \mathbb{N} \to 2^{\mathrm{AP}}$ を表す．任意の safety LTL 式 ϕ について，モニタとなる DFA M を生成できることが知られている [12]．M は 2^{AP} をアルファベットとし，$w \models \phi$ を満たす無限列 $w \in (2^{\mathrm{AP}})^\omega$ の接頭辞のみを受理し，それ以外の文字列を非受理とするような DFA である．

以下では DFA の記法として下記を用いる．DFA を 5 つ組 $(Q, \Sigma, \delta, q_0, F)$ として表し，Q を状態集合を表す有限集合，Σ を文字集合を表す有限集合，$\delta\colon Q \times \Sigma \to Q$ を遷移関数，$q_0 \in Q$ を始状態，$F \subset Q$ を終状態集合と呼ぶ．特に $\Sigma = \{0, 1\}$ のとき，このような DFA を 0/1 DFA と呼ぶ．状態 $q \in Q$ と文字列 $\mathbf{s} = \sigma_0\sigma_1\ldots\sigma_{n-1}$ について，$\delta(q, \mathbf{s}) := \delta(\ldots\delta(\delta(q, \sigma_0), \sigma_1), \ldots, \sigma_{n-1})$ と書く．DFA M と文字列 $\mathbf{s}$ について，M が s を受理する時 $M(\mathbf{s}) := 1$ とし，そうでなければ $M(\mathbf{s}) := 0$ と表記する．

2.3 オフラインアルゴリズム

TFHE を用いて DFA をオフラインに実行するアルゴリズムは，TFHE の原著論文 [4] で提案された．このアルゴリズムは入力として DFA M と，M への TRGSW の入力 $\mathrm{Enc}_{\mathrm{SK}}(\sigma_0), \ldots, \mathrm{Enc}_{\mathrm{SK}}(\sigma_{L-1})$ を受け取り，入力の受理・非受理を表す TLWE の暗号文を出力する．[4] では触れられていないが，適当な正整数 R_0 をとり，R_0 入力毎に内部状態の bootstrapping を行うことで，このアルゴリズムは入力サイズが大きい場合に拡張できる．また出力に対して bootstrapping を行うことで，動作させる DFA を秘匿できる．以下ではこの拡張されたアルゴリズムを指してオフラインアルゴリズムと呼ぶ.

このアルゴリズムでは入力である TRGSW の暗号文 $\mathrm{Enc}_{\mathrm{SK}}(\sigma_0), \ldots, \mathrm{Enc}_{\mathrm{SK}}(\sigma_{L-1})$ を**末尾から走査**する．そのため，アルゴリズムが動作する時点で M への入力が全て確定している必要があり，オンラインには実行できない．

3. プロトコル

データ提供者である Alice と，DFA の実行者である Bob が存在する．Alice はモニタリングを行う対象であるデータ系列 $\mathbf{s} = \sigma_0\sigma_1\ldots$ を持ち，Bob はモニタである DFA M を保持する．Alice は $M(\mathbf{s})$ を知りたいと思っているが，Bob から $\mathbf{s}$ 及び $M(\mathbf{s})$ を秘匿したい．一方で Bob は M を Alice から秘匿したい.

本研究では，Alice は malicious なエージェントであり，Bob は honest-but-curious なエージェントであると仮定する．すなわち，Alice はプロトコルに従わず，送受信データの改ざんを行うなどして Bob が持つ M の情報を知ろうとする．また Bob はプロトコルには従うものの，プロトコル中で自らが得た情報を用いて Alice の $\mathbf{s}$ 及び $M(\mathbf{s})$ を推

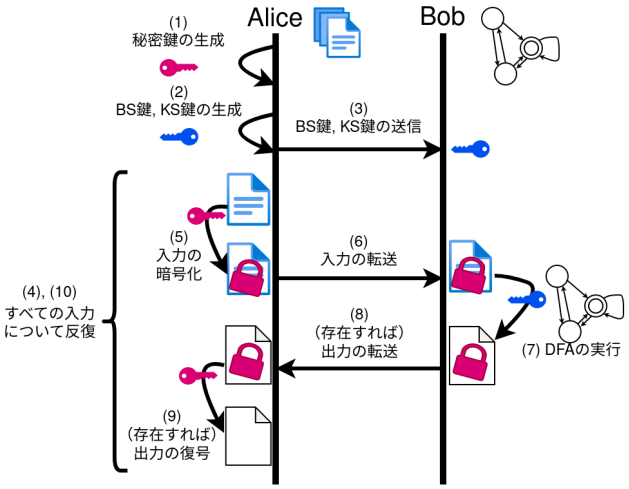

図 1 DFA をオンラインに実行する場合のプロトコル
Fig. 1 Protocol for executing a DFA in online setting

測しようとする．

M をオンラインに実行する場合，プロトコルは次のように定義される（**図 1**）．

(1) Alice は秘密鍵 SK を生成する．
(2) Alice は SK から bootstrapping 鍵 (BK)，及びアルゴリズム 2 を使用する場合は key switching 鍵 (KSK) を生成する．
(3) Alice は BK 及び KSK を Bob に送信する．
(4) $i \leftarrow 0$ とする．
(5) Alice は自らの入力 σ_i を SK を用いて暗号化し，暗号文 c_i を得る．
(6) Alice は c_i を Bob に送信する．
(7) Bob は，M，c_i，BK（及び KSK）を入力としてアルゴリズム 1 またはアルゴリズム 2 を実行する．
(8) アルゴリズムが出力を行う場合は，Bob はその出力である暗号文 d を Alice に送信する．d には bootstrapping が適用されている．
(9) d が存在する場合，Alice は d を SK を用いて復号し，結果 $M(\sigma_0 \dots \sigma_i)$ を得る．
(10) Alice がまだ送信すべき入力を持っている場合，$i \leftarrow i+1$ として (5) に遷移する．

3.1 プロトコルの安全性

Alice が自らの入力 $\mathbf{s}$ に対する正しい結果 $M(\mathbf{s})$ を得ることは，Bob が honest-but-curious であることから保証される．また Bob から $\mathbf{s}, M(\mathbf{s})$ が秘匿されることは TFHE の安全性から保証される．

各アルゴリズムでは，出力する暗号文に対して bootstrapping を適用してから出力する．そのため，出力となる暗号文を計算するためにアルゴリズム中で適用した TFHE の操作を Alice は知ることができない．すなわち，Bob が保有する M に関する情報を Alice は知ることができない．

また本プロトコルでは，いわゆる membership query のみを構成し，equivalence query は構成されない．また Alice は DFA の状態数を知りえない．一般に，状態数が明らかになっていない DFA を membership query のみで識別することはできないことが知られている [2] ため，Alice がプロトコルを繰り返すことで得た入出力のサンプルから M を推測することは困難であるといえる．

4. 反転アルゴリズム

オフラインアルゴリズムは DFA への入力を末尾から走査するため，実行時には入力が全て入手可能である必要があり，オンラインな DFA の実行には使用できない．そこで，DFA を**反転**させることで入力を末尾から受け取るような DFA を新たに構築し，この DFA をオフラインアルゴリズムを用いて実行することで，実効的に入力を先頭からオンラインに読み込むアルゴリズムを構成できる．

反転アルゴリズムの疑似コードをアルゴリズム 1 に掲げる．反転アルゴリズムでは DFA M' と，暗号文 $\mathrm{Enc}_{\mathrm{SK}}(\sigma_0), \mathrm{Enc}_{\mathrm{SK}}(\sigma_1), \dots$ が入力として与えられ，入力を R_2 個処理する毎に，その時点までに処理した入力を M' が受理するか否かを暗号文として出力する．本アルゴリズムではまず，DFA M' を反転させ NFA $\bar{M} = (Q', \Sigma, \bar{\delta}, F', \bar{F})$ を得る (1 行目)．ここで $\bar{\delta}(\bar{q}, \sigma) = \{q \mid \delta(q, \sigma) = \bar{q}\}$, $\bar{F} = \{q_0\}$ である．続いて冪集合構築を行い NFA $\bar{M}$ を等価な DFA M に変換する（2 行目）．一連の操作で得られた DFA M は，もともとの DFA M' に対する入力を末尾から与えることで M' と同じ受理・非受理結果が得られる．すなわち任意の入力文字列 $\mathbf{s} = \sigma_1\sigma_2 \dots \sigma_{L-1}$ について $\mathbf{s}^R = \sigma_{L-1} \dots \sigma_1$ とすると $M(\mathbf{s}^R) = M'(\mathbf{s})$ である．

続いて DFA M を，オンラインに与えられる入力を用いて実行する．3〜4 行目において初期設定を行う．このとき各 i について c_i は q_i が終了状態であれば $\textsc{Trivial}_1$ とし，そうでなければ $\textsc{Trivial}_0$ とする．その後 Alice からの入力を先頭から順に処理する（6〜19 行目）．処理する入力を $\mathrm{Enc}_{\mathrm{SK}}(\sigma_\alpha)$ とする．まず各状態 q_i について，$\mathrm{Enc}_{\mathrm{SK}}(\sigma_\alpha)$ に応じて $c_\delta(q_i, 1)$ または $c_\delta(q_i, 0)$ を選択し，結果を d_i に格納する（7〜8 行目）．その後，必要であれば後述する bootstrapping の処理を行い，得られた結果を対応する c_i に格納する．各入力 $\mathrm{Enc}_{\mathrm{SK}}(\sigma_\alpha)$ の処理が終わった時点で，c_i は $\delta(q_i, \sigma_\alpha\sigma_{\alpha-1} \dots \sigma_0)$ が終了状態であるか否かを表す TRLWE の暗号文になっている．特に c_0 は M が $\sigma_\alpha\sigma_{\alpha-1} \dots \sigma_0$ を受理するか，すなわち M' が $\sigma_0 \dots \sigma_\alpha$ を受理するかを表す．

$\textsc{CMux}$ が出力する暗号文のノイズは入力のそれよりも大きいため，各繰り返し毎に c_i のノイズは増大する．そのため入力長が大きい場合，暗号文 c_i に対して bootstrapping を行いそのノイズを減らす必要がある（9〜12 行目）．そのため，前回の bootstrapping（あるいはアルゴリズムの開

アルゴリズム 1 反転アルゴリズム

入力: 0/1 DFA $M' = (Q', \Sigma = \{0,1\}, \delta', q'_0, F')$
入力: オンラインに与えられる TRGSW の暗号文 $\mathrm{Enc}_{\mathrm{SK}}(\sigma_0), \mathrm{Enc}_{\mathrm{SK}}(\sigma_1), \ldots$（ここで $\sigma_i \in \Sigma$）
入力: bootstrapping 鍵 BK
入力: bootstrapping を行う頻度を表す自然数 R_1
入力: 出力を行う頻度を表す自然数 R_2
出力: 各正整数 i について，$M'(q'_0, \sigma_0\sigma_1 \ldots \sigma_{iR_2})$ を表す TLWE の暗号文

1: M' を反転させ NFA $\bar{M} = (Q', \Sigma, \bar{\delta}, F', \bar{F})$ を作る
2: $\bar{M}$ について冪集合構築を行い等価な 0/1 DFA $M = (Q = \{q_i\}_{i=0}^{|Q|-1}, \Sigma, \delta, q_0, F)$ を作る
3: **for** $i = 0, 1, \ldots, |Q|-1$ **do**
4: $\quad c_i \leftarrow q_i \in F$? $\textsc{Trivial}_1$: $\textsc{Trivial}_0$
5: $\alpha \leftarrow 0$
6: **while** 入力 $\mathrm{Enc}_{\mathrm{SK}}(\sigma_\alpha)$ が存在する **do**
7: $\quad$ **for** $i = 0, 1, \ldots, |Q|-1$ **do**
8: $\quad\quad d_i \leftarrow \textsc{CMux}(\mathrm{Enc}_{\mathrm{SK}}(\sigma_\alpha), c_{\delta(q_i,1)}, c_{\delta(q_i,0)})$
9: $\quad$ **if** $\alpha \neq 0 \wedge \alpha \mod R_1 = 0$ **then**
10: $\quad\quad$ **for** $i = 0, 1, \ldots, |Q|-1$ **do**
11: $\quad\quad\quad e_i \leftarrow \textsc{SampleExtract}_0(d_i)$
12: $\quad\quad\quad d_i \leftarrow \textsc{Bootstrapping}_{\mathrm{BK}}(e_i)$
13: $\quad \mathbf{c} \leftarrow \mathbf{d}$
14: $\quad$ **if** $\alpha \neq 0 \wedge \alpha \mod R_2 = 0$ **then**
15: $\quad\quad t_0 \leftarrow \textsc{SampleExtract}_0(c_0)$
16: $\quad\quad t_1 \leftarrow \textsc{Bootstrapping}_{\mathrm{BK}}(t_0)$
17: $\quad\quad t_2 \leftarrow \textsc{SampleExtract}_0(t_1)$
18: $\quad\quad t_2$ を出力する
19: $\quad \alpha \leftarrow \alpha + 1$

始）から処理した入力の個数がしきい値 R_1 を超えた場合に bootstrapping 処理を行う．

出力を行う場合は適用した加算及び CMux の回数が出力から漏れることを防ぐために，c_0 に bootstrapping を適用したものをこのアルゴリズムの出力とする（14〜18 行目）．

アルゴリズム 1 の動作時間と使用メモリ量は，反転後の $|Q|$ について線形である．冪集合構築を行って得られる DFA の状態数は，最悪の場合，入力した NFA の状態数の指数倍に増大するため，この場合，反転アルゴリズムは極めて低速になり，かつ多くのメモリを消費しうる．

5. Forward LUT アルゴリズム

Foward LUT（FLUT）アルゴリズムでは，DFA の現状態を one-hot ベクトルを表す暗号文として表現し，Alice から与えられる各入力により起こる状態の遷移を TFHE で可能な操作を用いて暗号文上で行うことで，オンラインに DFA を実行する．反転アルゴリズムと異なりこの手法では DFA の変更が必要ないため，反転アルゴリズムでは実行困難な DFA でも FLUT アルゴリズムによって実行できる場合がある．

FLUT アルゴリズムでは，DFA の現状態は $|Q|$ 個の TRLWE の暗号文 $\{c_i\}_{i=0}^{|Q|-1}$ として表現される．これらの暗号文は，定数項が 0 または 1 で，残りの項が 0 であるような多項式を表す．現状態が q_i のとき，c_i の定数項が 1 であり，残りの項は全て 0 である．以下ではこの形式を**現状態形式 1** として参照する．

また，アルゴリズムの一部では，現状態を表す別の形式として，現状態が q_i のときに，X^i の係数のみが 1 であり，残りの項が 0 であるような多項式を表す TRLWE の暗号文を用いる．以下ではこの形式を**現状態形式 2** として参照する．現状態形式 1 では $|Q|$ 個の暗号文を用いたのに対して，現状態形式 2 では 1 個の暗号文で現状態を表現している．なお TRLWE の暗号文が表す多項式は N 個の係数しか持たないため，現状態形式 2 を用いるためには $|Q| \leq N$ を満たす必要がある．ただし $|Q| > N$ の場合でも，$\lceil |Q|/N \rceil$ 個の TRLWE の暗号文を用いるようにアルゴリズムを容易に拡張できる．また，TRLWE に保存するべき状態の数を $|Q|$ よりも減らせる場合，後述するように，$|Q| > N$ であっても 1 個の TRLWE の暗号文で現状態形式 2 を使用できる．以下では簡単のため $|Q| \leq N$ を仮定する．

FLUT アルゴリズムの疑似コードをアルゴリズム 2 に掲げる．アルゴリズムは入力として実行するべき 0/1 DFA M と M への入力，及び bootstrapping 鍵と key switching 鍵を受け取る．さらにパラメタとして，正整数 K と R_3 を受け取り，各々一度に読む入力の個数と，出力の頻度を表す．FLUT アルゴリズムは入力を K 個まとめて処理するため，出力は K の倍数個の入力を読み込んだ時点でのみ可能である．したがって R_3 は K の倍数である必要がある．アルゴリズムの出力は正整数 i について，$M(q_0, \sigma_0\sigma_1 \ldots \sigma_{iR_3})$ を表す TLWE の暗号文である．

アルゴリズムでは，まず初期設定を行う（1〜3 行目）．初期状態である q_0 に対応する c_0 に $\textsc{Trivial}_1$ を代入し，それ以外の状態に $\textsc{Trivial}_0$ を代入することで，現状態を q_0 に設定する．

続いて入力を処理するループを実行する（5〜21 行目）．ループは Step 1，2，3 と，出力処理によって構成される．このループでは，Alice からの入力を K 個まとめて処理を行うことで Step 3 での bootstrapping 処理の回数を K 入力おきにし，パフォーマンスを改善する．

Step 1 では，あり得る 2^K 通りの入力の各々について，現在の状態からその入力が与えられた場合に遷移する状態を現状態形式 2 の暗号文として計算する（6〜9 行目）．すなわち現状態が q_i で入力 $\mathbf{s}$ が与えられる場合には，$\delta(q_i, \mathbf{s})$ を表すような TRLWE の暗号文 $d_{\mathbf{s}}$ を生成する．最終的に 2^K 個の TRLWE $d_0, d_1, \ldots, d_{2^K-1}$ が得られる．

Step 2 では，Alice から与えられた K 個の入力を用いて $\textsc{Lookup}$ を呼び出し，$d_0, d_1, \ldots, d_{2^K-1}$ の中から正しい入力での遷移結果を選択する（10 行目）．2.1 項で示したように，$\textsc{Lookup}$ 中では 2^K 回の CMux の適用によりこれが実現される．$\textsc{Lookup}$ の結果得られた TRLWE の暗号文

を d とする．Step 1 と 2 の一連の動作は，Step 1 で lookup table（LUT）のエントリを作成し，Step 2 で Alice の入力を用いて LUT の実行を行ったと見ることができる．

Step 3 では，得られた d から現状態形式 1 を満たす $|Q|$ 個の TRLWE 暗号文を復元し c_i に代入する（11〜15 行目）．d は現状態形式 2 を満たすため，まず SampleExtract を用いて各 X^i の係数の値（q_i が現状態か否か）を TLWE の暗号文として取り出す．続いて，操作を通じて蓄積したノイズを Bootstrapping を用いて除去する．その後 SampleExtract と TLWE-to-TRLWE を用いて定数項に情報が乗った TRLWE の暗号文に変換する．

最後に，出力が要求される場合は，出力を行う（17〜21 行目）．出力では，まず $\sum_{q_i \in F} c_i$ を計算する．現状態形式 1 を満たす暗号文 $c_0, \ldots, c_{|Q|-1}$ のうち，定数項に 1 が入っているのは唯一つであるため，加算の結果得られる TRLWE の暗号文は，現状態が終状態であれば定数項に 1 を持ち，そうでなければ 0 を持つ．Bob の操作を秘匿するために Bootstrapping を実行し，得られた TLWE の暗号文を出力とする．

FLUT アルゴリズムでは次のような高速化が可能である．一般に k を正整数として $k|Q| \leq N$ のとき，1 つの TRLWE の暗号文に k 個の入力パターンに関する情報を保存できる．これを利用して，Alice の入力を 2 段階に分けて読み込むことで CMux を実行する回数を減らすことができる．また，各繰り返しにおいて，その時点で到達しうる状態集合 S を管理しておき，S に含まれる状態のみについてアルゴリズム中で考慮することで，実効的な $|Q|$ を減らすことができる．

アルゴリズム 2 FLUT アルゴリズム

入力: 0/1 DFA $M = (Q = \{q_i\}_{i=0}^{|Q|-1}, \Sigma = \{0,1\}, \delta, q_0, F)$
入力: オンラインに与えられる TRGSW の暗号文 $\mathrm{Enc}_{\mathrm{SK}}(\sigma_0), \mathrm{Enc}_{\mathrm{SK}}(\sigma_1), \ldots$（ここで $\sigma_i \in \Sigma$）
入力: bootstrapping 鍵 BK
入力: key switching 鍵 KSK
入力: 一度に読む入力個数の最大値 K
入力: 出力を行う頻度を表す自然数 R_3（R_3 は K の倍数）
出力: 正整数 i について，$M(q_0, \sigma_0\sigma_1 \ldots \sigma_{iR_3})$ を表す TLWE の暗号文

1: $c_0 \leftarrow \mathrm{Trivial}_1$ ▷ 初期設定
2: **for** $i = 1, 2, \ldots, |Q| - 1$ **do**
3: $c_i \leftarrow \mathrm{Trivial}_0$
4: $\alpha \leftarrow 0$
5: **while** 入力 $\{\mathrm{Enc}_{\mathrm{SK}}(\sigma_{\alpha+i})\}_{i=0}^{K-1}$ が存在する **do**
6: **for** $\mathbf{s} \in \{0,1\}^K$ **do** ▷ Step 1
7: $d_{\mathbf{s}} \leftarrow \mathrm{Trivial}_0$
8: **for** $i = 0, 1, \ldots, |Q| - 1$ **do**
9: $q_j \leftarrow \delta(q_i, \mathbf{s})$; $d_{\mathbf{s}} \leftarrow d_{\mathbf{s}} + \mathrm{CyclicShift}_j(c_i)$
10: $d \leftarrow \mathrm{Lookup}_{d_0,\ldots,d_{2^K-1}}(\mathrm{Enc}_{\mathrm{SK}}(\sigma_s), \ldots, \mathrm{Enc}_{\mathrm{SK}}(\sigma_{s+(K-1)}))$ ▷ Step 2
11: **for** $i = 0, 1, \ldots, |Q| - 1$ **do** ▷ Step 3
12: $t_0^i \leftarrow \mathrm{SampleExtract}_i(d)$
13: $t_1^i \leftarrow \mathrm{Bootstrapping}_{\mathrm{BK}}(t_0^i)$
14: $t_2^i \leftarrow \mathrm{SampleExtract}_0(t_1^i)$
15: $c_i \leftarrow \text{TLWE-to-TRLWE}_{\mathrm{KSK}}(t_2^i)$
16: $\alpha \leftarrow \alpha + K$
17: **if** $\alpha \mod R_3 = 0$ **then** ▷ 出力
18: $t_0 \leftarrow \mathrm{SampleExtract}_0(\sum_{q_i \in F} c_i)$
19: $t_1 \leftarrow \mathrm{Bootstrapping}_{\mathrm{BK}}(t_0)$
20: $t_2 \leftarrow \mathrm{SampleExtract}_0(t_1)$
21: t_2 を出力とする

6. 実験

6.1 実験設定

本研究で提案した手法の評価を，1 型糖尿病患者の血糖値のシミュレーションデータと，そのモニタリングを行う LTL 式を用いて行った．実験はオフライン・反転・FLUT アルゴリズムの 3 手法で行った．各アルゴリズムにおけるパラメタは $R_0 = 30000, R_1 = 30000, R_2 = 30, K = 15, R_3 = 30$ とした．各入力において到達しうる状態数が 256 以下の場合，各入力を処理した結果得られる結果が誤る確率は 2^{-32} 以下である．オンラインアルゴリズムはどちらも，30 入力毎に 1 つの出力を行うパラメタである．また TFHE のパラメタである N は 1024 とした．

実験には CPU として Intel Xeon Silver 4216 (32C64T @ 3.2GHz) を用い，メモリは 128GB，OS は Ubuntu 20.04.2 LTS を使用した．

本実験では，血糖値データのモニタリング仕様として [3] における ψ_1, ψ_2, ψ_4，及び [14] における $\phi_1, \phi_2, \phi_4, \phi_5$ を，LTL によって記述したものを用いた（表 1）．なお [14] における ϕ_4, ϕ_5 の定義にはデータに依存して定まるパラメタが存在するが，ここでは適当な定数とした．

表 1 実験において使用した LTL 式
Table 1 LTL formulae used in our experiments

変数名		LTL 式
[3]	ψ_1	$\mathsf{G}_{[100,700]}(p_5 \vee p_4 \vee p_3 \vee (p_2 \wedge p_1 \wedge p_0))$
	ψ_2	$\mathsf{G}_{[100,700]}(\neg p_5 \vee (\neg p_2 \wedge \neg p_3 \wedge \neg p_4 \wedge p_5))$
	ψ_4	$\mathsf{G}_{[600,700]}((\neg p_5 \wedge \neg p_5) \vee (\neg p_2 \wedge \neg p_3 \wedge p_4 \wedge \neg p_5) \vee (\neg p_0 \wedge \neg p_1 \wedge p_2 \wedge \neg p_3 \wedge p_4 \wedge \neg p_5))$
[14]	ϕ_1	$\mathsf{G}((p_0 \wedge p_1 \wedge p_2 \wedge \neg p_3 \wedge \neg p_4 \wedge \neg p_5) \vee (p_3 \wedge \neg p_4 \wedge \neg p_5) \vee (\neg p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge p_4 \wedge \neg p_5))$
	ϕ_2	$\mathsf{G}((\neg p_0 \wedge \neg p_2 \wedge \neg p_3 \wedge \neg p_4 \wedge \neg p_5) \vee (\neg p_1 \wedge \neg p_2 \wedge \neg p_3 \wedge \neg p_4 \wedge \neg p_5) \vee (p_0 \wedge p_1 \wedge p_3 \wedge p_4 \wedge \neg p_5) \vee (p_2 \wedge p_3 \wedge p_4 \wedge \neg p_5))$
	ϕ_4	$\mathsf{G}(((\neg p_5 \wedge \neg p_4 \wedge \neg p_3) \wedge ((p_2 \wedge \neg p_1) \vee \neg p_2)) \implies \mathsf{F}_{[0,25]}((p_2 \wedge p_1) \vee p_3 \vee p_4 \vee p_5))$
	ϕ_5	$\mathsf{G}(p_5 \vee (p_4 \wedge p_2) \vee (p_4 \wedge p_3) \implies \mathsf{F}_{[0,25]}((\neg p_5 \wedge p_4 \wedge \neg p_3 \wedge \neg p_2) \vee (\neg p_5 \wedge \neg p_4)))$

LTL 式のモニタへの変換は Spot [7] を利用して行った．ここで，本研究で提示したアルゴリズムは全て 0/1 DFA を必要とする一方で，Spot が与えるモニタの文字集合は 2^{AP} である．実験では AP に添え字を振り，その順に 0 ま

表 2 各 LTL 式を変換した DFA の状態数と，反転した DFA の状態数．ψ_4 に対応する反転後の DFA は状態数が極めて大きく，構成できなかった．

Table 2 The size of the original and the reversed DFAs obtained by converting each LTL formula. The reversed DFA for ψ_4 is too huge and its construction aborted.

LTL 式	状態数	反転時の状態数
ψ_1	7215	1856014
ψ_2	6015	1531414
ψ_4	4617	—
ϕ_1	13	13
ϕ_2	13	13
ϕ_4	186	186
ϕ_5	234	234

たは 1 を並べることで入力をエンコードした．すなわち，入力 $\sigma \in 2^{\mathrm{AP}}$ を $\sigma' \in \{0,1\}^{|\mathrm{AP}|}$ に変換した．またモニタも，遷移の途中に補助的な状態を追加することによって，同様のエンコーディングに従い入力列を受け取るような 0/1 DFA に変換した．この変換によって得られた DFA の状態数と，その DFA を反転し得られた DFA の状態数を表 2 に示す．$\phi_1, \phi_2, \phi_4, \phi_5$ は反転を行っても DFA の状態数がほとんど変化しない一方で，ψ_1, ψ_2, ψ_4 は反転を行うと DFA の状態数が極めて増加する．特に ψ_4 については反転後の状態数が極めて大きく，その DFA を計算することができなかった．

1 型糖尿病患者の血糖値データのシミュレーションには，UVA/Padova Type 1 Diabetes Simulator [10] の Python 実装である simglucose [13] を使用した．シミュレーションデータは 1 分間隔での測定とし，ψ_1, ψ_2, ψ_4 では午後 7 時から午前 7 時までの 720 分間，$\phi_1, \phi_2, \phi_4, \phi_5$ では 7 日間測定したデータを使用した．どちらのデータも計測値は 6bit に丸め，6 つの AP によってエンコードした．

6.2 実験結果及び議論

実験結果を表 3 に示す．各実行時間は Bob が DFA を評価するためにかかった時間を表しており，10 回の試行の平均と標準偏差を示している．ただし ψ_4 は反転アルゴリズムが適用できなかったため，その実行時間は計測されていない．なお，この実行時間には出力のための bootstrapping 処理の時間を含む．

実験結果からは，どのケースでも反転・FLUT アルゴリズムの双方が各入力 bit を平均 100 ミリ秒以内に処理していることが分かる．計測値は 6bit で表されるため，各計測値は平均 600 ミリ秒以内に処理される．今回の計測値は 1 分おきに測定されるため，どのケースでも十分な速度でデータの処理を行えることが分かる．

また，いずれの計測結果でも，反転アルゴリズムと FLUT アルゴリズムはオフラインアルゴリズムよりも低速である．

表 3 実験結果．試行回数は 10．ψ_4 のアルゴリズム 1 は，対応する DFA の反転が不可能であったため計測できなかった．

Table 3 Summary of the experiment results of 10 executions. The result for ψ_4 of Algorithm 1 is missing since we could not construct its reversed DFA.

LTL 式	入力サイズ（bit）	アルゴリズム	実行時間（秒）	入力 bit 毎の平均実行時間（ミリ秒）
ψ_1	4326	オフライン	0.86 ± 0.03	0.20 ± 0.01
		反転	333.28 ± 11.09	77.04 ± 2.56
		FLUT	16.27 ± 0.30	3.76 ± 0.07
ψ_2	4326	オフライン	0.74 ± 0.07	0.17 ± 0.02
		反転	271.18 ± 1.35	62.69 ± 0.31
		FLUT	16.11 ± 0.14	3.72 ± 0.03
ψ_4	4326	オフライン	0.41 ± 0.02	0.09 ± 0.00
		反転	—	—
		FLUT	12.96 ± 0.24	3.00 ± 0.05
ϕ_1	60486	オフライン	6.42 ± 0.09	0.11 ± 0.00
		反転	48.67 ± 0.87	0.80 ± 0.01
		FLUT	212.28 ± 1.41	3.51 ± 0.02
ϕ_2	60480	オフライン	5.84 ± 0.10	0.10 ± 0.00
		反転	48.60 ± 0.77	0.80 ± 0.01
		FLUT	211.72 ± 2.20	3.50 ± 0.04
ϕ_4	60486	オフライン	16.44 ± 0.14	0.27 ± 0.00
		反転	58.26 ± 0.95	0.96 ± 0.02
		FLUT	396.02 ± 7.95	6.55 ± 0.13
ϕ_5	60486	オフライン	17.37 ± 0.15	0.29 ± 0.00
		反転	59.00 ± 0.58	0.98 ± 0.01
		FLUT	476.68 ± 7.11	7.88 ± 0.12

これは，オフラインアルゴリズムでは出力を一度しか行わないのに対して，オンラインアルゴリズムでは複数回（今回の設定では 30 入力に 1 度）行う必要があることに起因すると考えられる．出力の際には bootstrapping 処理を行う必要があるため，出力が多く行われるほど低速になる．

実験結果からは，反転アルゴリズムと FLUT アルゴリズムのどちらが高速かは，動作させる LTL 式によって大きく異なることが示唆される．動作させる DFA を反転させても状態数が増加しない場合（$\phi_1, \phi_2, \phi_4, \phi_5$），反転アルゴリズムは FLUT アルゴリズムよりも高速に動作する．これは，反転アルゴリズムでは $R_1 = 30,000$ 入力毎に bootstrapping すればよいのに対して，FLUT では $K = 15$ 入力毎に bootstrapping しなければならないことに由来すると考えられる．一方で，ψ_1, ψ_2, ψ_4 では反転後の DFA の状態数が大きく増加するため，FLUT アルゴリズムのほうが反転アルゴリズムよりも高速に動作している．特に ψ_4 では状態数が極めて増加するため反転アルゴリズムによる実験を行うことができなかったが，FLUT アルゴリズムでは動作させることができた．またこれらの LTL 式では反転前から状態数は $N = 1024$ を超えているが，各入力において到達可能な状態の数はいずれも 3 以下であるため，

FLUT アルゴリズムの適用が可能であった．

なお，メモリ使用量はほとんどのケースで数百 MB〜数 GB 程度であった．ただし ψ_1, ψ_2 への反転アルゴリズムの適用時は，状態数が非常に多いため，50GB〜60GB 程度のメモリ使用量となった．

7. 関連研究

データを秘匿して LTL によるモニタリングを行う手法を提案している文献は，我々の知る限り [1] のみである．[1] では LTL 式を DFA に変換し，この DFA を Private Information Retrieval 及び Private Set Intersection と呼ばれる手法を用いて実行する．[1] ではオフライン・オンラインの両方の設定を扱っている．[1] と提案手法との大きな違いは，そのセキュリティモデルである．[1] では，Alice は Bob が動作させる DFA の定義を予め知る必要がある．特にオンラインアルゴリズムでは，各入力に応じて現状態を Alice が把握する必要がある．一方で，提案手法では Bob が動作させている DFA は Alice から秘匿され，DFA の状態数や現状態を Alice は知ることができない．また [1] はアルゴリズムの提案にとどまり，その実装・評価を行っていないが，本研究では TFHE を用いて実装・評価を行い，実用的に使用できる速度で動作することを確認した．

Oblivious DFA evaluation [11] は，garbled circuit を拡張した手法を用いて，Alice が持つ入力データと Bob が持つ DFA を秘匿しながら DFA を実行する．この手法では，Bob は自らが持つ DFA を garble し，それを Alice に送信する．Alice は Bob から送られてきた garbled DFA と紛失通信によって得た入力に対応する暗号文を用いて DFA を実行する．提案手法との大きな違いとして，[11] では Bob が実行する DFA の状態数が Alice に漏洩するという問題がある．また [11] ではオフラインの設定のみを扱っている．

8. 結論

本研究では Alice が持つデータに対して Bob が LTL を用いたオンラインモニタリングを行う際に，Bob から Alice のデータとモニタリング結果を秘匿し，Alice から Bob の LTL 式を秘匿するような秘匿 LTL オンラインモニタリングを行うためのプロトコルを提案した．プロトコルを遂行するための手法として TFHE を用いた反転アルゴリズムと FLUT アルゴリズムを提案し，ケーススタディとして血糖値のシミュレーションデータと監視仕様を用いてその実用性を検証した．

謝辞 本研究は，JST，CREST，JPMJCR2012，JPMJCR19K5，及び JSPS 科研費 20K21793 の支援を受けたものである．

参考文献

[1] Houssam Abbas. Private runtime verification: work-in-progress. In *Proc. EMSOFT'19*, pp. 1–2, New York, New York, 2019. ACM Press.

[2] Dana Angluin. A note on the number of queries needed to identify regular languages. *Information and Control*, Vol. 51, No. 1, pp. 76–87, October 1981.

[3] Fraser Cameron, Georgios Fainekos, David M. Maahs, and Sriram Sankaranarayanan. Towards a Verified Artificial Pancreas: Challenges and Solutions for Runtime Verification. In Ezio Bartocci and Rupak Majumdar, editors, *Runtime Verification*, Vol. 9333 of *LNCS*, pp. 3–17. Springer International Publishing, Cham, 2015.

[4] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology*, Vol. 33, No. 1, pp. 34–91, January 2020.

[5] Stéphane Demri and Paul Gastin. Specification and Verification using Temporal Logics. In *Modern Applications of Automata Theory*, pp. 457–493. Co-Published with Indian Institute of Science (IISc), Bangalore, India, 2012.

[6] Dimiter V. Dimitrov. Medical Internet of Things and Big Data in Healthcare. *Healthcare Informatics Research*, Vol. 22, No. 3, pp. 156–163, July 2016.

[7] Alexandre Duret-Lutz, Alexandre Lewkowicz, Amaury Fauchille, Thibaud Michaud, Etienne Renault, and Laurent Xu. Spot 2.0 — a framework for LTL and ω-automata manipulation. In *Proc. ATVA'16*, Vol. 9938 of *LNCS*, pp. 122–129. Springer, October 2016.

[8] Craig Gentry. *A fully homomorphic encryption scheme*. phd, Stanford University, Stanford, CA, USA, 2009.

[9] Orna Kupferman and Moshe Y. Vardi. Model Checking of Safety Properties. *Formal Methods in System Design*, Vol. 19, No. 3, pp. 291–314, November 2001.

[10] Chiara Dalla Man, Francesco Micheletto, Dayu Lv, Marc Breton, Boris Kovatchev, and Claudio Cobelli. The UVA/PADOVA Type 1 Diabetes Simulator. *Journal of Diabetes Science and Technology*, Vol. 8, No. 1, pp. 26–34, January 2014.

[11] Payman Mohassel, Salman Niksefat, Saeed Sadeghian, and Babak Sadeghiyan. An Efficient Protocol for Oblivious DFA Evaluation and Applications. In Orr Dunkelman, editor, *Proc. CT-RSA'12*, Vol. 7178 of *LNCS*, pp. 398–415. Springer, Berlin, Heidelberg, 2012.

[12] Deian Tabakov and Moshe Vardi. Optimized Temporal Monitors for SystemC. Vol. 41, pp. 436–451, 2010.

[13] Jinyu Xie. Simglucose v0.2.1. `https://github.com/jxx123/simglucose`, 2018 (accessed on: 2021-08-10).

[14] William Young, John Corbett, Matthew S. Gerber, Stephen Patek, and Lu Feng. DAMON: A Data Authenticity Monitoring System for Diabetes Management. In *Proc. IoTDI'18*, pp. 25–36, Orlando, FL, 2018. IEEE.