Verschlüsselt Rechnen: Sichere Verarbeitung Verschlüsselter Medizinischer Daten am Beispiel der Klassifikation von EKG Daten

Ahmad-Reza Sadeghi, Thomas Schneider

{ahmad.sadeghi,thomas.schneider}@trust.rub.de

Abstract: Die rasant fortschreitende Modernisierung des Gesundheitswesens durch neuartige elektronische Dienste wie die elektronische Patientenakte und medizinische online Dienste erlaubt es, medizinische Prozesse effizienter zu gestalten. Andererseits birgt die digitale Verarbeitung sensibler Patientendaten Risiken und Gefahren hinsichtlich des Datenschutzes und des Missbrauchs.

Klassische kryptographische und IT sicherheitstechnische Methoden erlauben die sichere Verteilung und Speicherung medizinischer Daten. Allerdings erfordert die Verarbeitung der Daten deren Entschlüsselung. Zu diesem Zeitpunkt kann auf die Daten im Klartext (z.B. durch Systemadministratoren) zugegriffen und die Vertraulichkeit verletzt werden. Um dies zu verhindern, sollten Daten in verschlüsselter Form verarbeitet werden. Für ebendieses "Rechnen unter Verschlüsselung" wurden in den vergangenen 25 Jahren verschiedene kryptographische Verfahren vorgeschlagen.

Die Ziele dieser Arbeit sind folgende: 1) Der aktuelle Stand der Forschung im Bereich des effizienten Rechnens unter Verschlüsselung wird zusammengefasst. 2) Ein Werkzeug wird vorgestellt, das es erlaubt, effiziente Protokolle zum Rechnen unter Verschlüsselung abstrakt und ohne detaillierte kryptographische Kenntnisse zu beschreiben und automatisch zu generieren. 3) Es wird gezeigt, wie das vorgestellte Werkzeug exemplarisch zum Generieren eines medizinischen Web-Services verwendet werden kann, der EKG Daten in verschlüsselter Form klassifiziert.

1 Einleitung

Die Gesundheitsindustrie entwickelt in zunehmendem Maße medizinische Online-Dienste für Patienten, die ihnen schnell Auskunft über ihren Gesundheitszustand geben und gegebenenfalls weitere Maßnahmen wie das Aufsuchen eines Facharztes empfehlen können. Solche Technologien haben das Potential, medizinische Daten und Wissen für Millionen von Benutzern weltweit zu verwalten, verarbeiten, speichern, verteilen und allgegenwärtig zur Verfügung zu stellen. Prominente Beispiele für solche Online-Dienste sind Google Health [McB08] und Microsoft HealthVault [Bla08]. Da in diesen Diensten sensible Patientendaten verarbeitet werden, ist der Datenschutz von höchster Priorität. Falls es Zweifel seitens der potentiellen Nutzer an diesem Schutz gibt, wird die Verbreitung solcher neuer elektronischer Gesundheitsdienste verhindert oder zumindest verlangsamt. Auch großflächige Systeme zum Speichern von medizinischen Daten wie die elektronische Patientenakte könnten dahingehend erweitert werden, dass sie die vertrauliche Verarbeitung

medizinischer Daten ermöglichen.

Methoden der klassischen Kryptographie wie symmetrische und asymmetrische Verschlüsselung erlauben den sicheren Transport von Daten zwischen Rechnern und bieten Schutz gegen Angreifer von außen und unberechtigte Innentäter. Diese Methoden erlauben es, medizinische Daten sicher auf einen oder mehrere Server zu verteilen und dort verschlüsselt zu speichern. Zur Verarbeitung solcher verschlüsselter Daten müssten diese jedoch zunächst entschlüsselt und dann unverschlüsselt verarbeitet werden, bevor das Ergebnis wieder verschlüsselt wird. Zu dem Zeitpunkt, an dem die Daten unverschlüsselt vorliegen, könnten Innentäter wie zum Beispiel Systemadministratoren die Daten ausspionieren und somit die Vertraulichkeit des Systems verletzen. Um dies zu verhindern, sollten Daten in verschlüsselter Form verarbeitet werden, ohne sie zu entschlüsseln. Für ebendieses "Rechnen unter Verschlüsselung" wurden in den vergangenen 25 Jahren verschiedene kryptographische Verfahren vorgeschlagen.

1.1 Inhalt

In dieser Arbeit fassen wir zunächst in §2 den aktuellen Stand der Technik zum Rechnen unter Verschlüssung in, auch ohne kryptographischen Hintergrund, leicht verständlicher Form zusammen. Unsere Beschreibung basiert auf dem Modell von [KSS10], in dem Algorithmen zum Rechnen unter Verschlüsselung programmiert werden können, wobei von der zugrunde liegenden Kryptographie abstrahiert wird.

Anschließend stellen wir in §3 das zugehörige Werkzeug TASTY (Tool for Automating Secure Two-partY computations) [HKS+10] vor, das es Programmierern erlaubt, solche Algorithmen zum Rechnen unter Verschlüsselung in einer domänen-spezifischen Hochsprache zu beschreiben und mit einem Compiler automatisch in effiziente und sichere kryptographische Protokolle zu übersetzen.

Als Anwendung betrachten wir in §4 einen medizinischen Web-Service, der EKG Daten in verschlüsselter Form klassifiziert. Hierbei wird sowohl die Privatsphäre des Patienten als auch das geistige Eigentum des Dienstanbieters geschützt. Wir zeigen wie unter Verwendung von TASTY die diesem Web-Service zu Grunde liegenden kryptographischen Protokolle beschrieben und automatisch generiert werden können.

Abschließend diskutieren wir in §5 den beschriebenen Ansatz des Rechnens auf verschlüsselten Daten im Kontext medizinischer Anwendungen.

2 Verfahren zum effizienten Rechnen unter Verschlüsselung

Beim *Rechnen unter Verschlüsselung* wollen zwei Parteien, Client $\mathcal{C}$ und Server $\mathcal{S}$, eine beiden bekannte Funktion f auf ihren geheimen Eingaben x bzw. y so berechnen, dass lediglich das Ergebnis $f(x, y)$ bekannt wird, jedoch keine zusätzliche Information über die Eingabe der anderen Partei oder Zwischenergebnisse.

Sicherheitsmodell. Im Folgenden nehmen wir an, dass beide Parteien ehrlich aber neugierig sind (engl. honest-but-curious). Sie verhalten sich in dem Sinne *ehrlich*, daß sie die im Protokoll vorgegebenen Aktionen durchführen, sind aber *neugierig*, da sie versuchen, aus den empfangenen Nachrichten zusätzliche Informationen zu rekonstruieren. Die von uns vorgestellten Protokolle garantieren, dass falls sich beide Parteien ehrlich verhalten, sie keine zusätzlichen Informationen über die Eingabe der anderen Partei oder Zwischenergebnisse lernen. Dieses Modell ist nicht trivial und verhindert bereits viele realistische Angriffsszenarien wie "Insider Attacken", bei denen zum Beispiel Systemadministratoren Zugang zu geheimen Informationen erhalten. Für eine Übersicht zu erweiterten Protokollen, die sicher gegen stärkere Angreifer sind, verweisen wir auf [KSS10].

Ansätze. Es gibt zwei prinzipielle Ansätze zum Rechnen unter Verschlüsselung:

Beim *"Rechnen mit verschlüsselten Funktionen"* [Yao86] wird zusätzlich zu den verschlüsselten Daten, für jede der zu berechnenden Elementaroperationen eine verschlüsselte Übersetzungstabelle verwendet, die es erlaubt, eine Funktion *einmalig* auf den verschlüsselten Daten zu berechnen.

Beim *"Rechnen auf verschlüsselten Daten"* [Pai99] werden die Daten mit einem speziellen Verschlüsselungsverfahren, sogenannter "homomorpher Verschlüsselung" verschlüsselt, das es erlaubt, ohne Verwendung von Übersetzungstabellen, bestimmte Operationen auf den verschlüsselten Daten mehrmals zu berechnen.

Im Folgenden stellen wir zunächst effiziente Verfahren für diese beiden Ansätze vor und beschreiben anschließend ein Modell, das deren beliebige Kombination erlaubt.

2.1 Rechnen auf verschlüsselten Daten

Homomorphe Verschlüsselungsverfahren erlauben es, bestimmte Operationen auf verschlüsselten Daten durchzuführen.

Dies erlaubt das *Rechnen auf verschlüsselten Daten*: Client $\mathcal{C}$ generiert ein Schlüsselpaar für ein homomorphes Verschlüsselungsverfahren und schickt den öffentlichen Schlüssel gemeinsam mit der homomorphen Verschlüsselung seiner Inputs an $\mathcal{S}$. Dieser kann dank der homomorphen Eigenschaften auf den (homomorph) verschlüsselten Daten die gewünschte Funktion berechnen und schickt das verschlüsselte Ergebnis zurück an $\mathcal{C}$, der entschlüsselt.

Wir schreiben $[\![x]\!]$ für die homomorphe Verschlüsselung (Chiffretext) des Klartextes x unter $\mathcal{C}$'s öffentlichem Schlüssel.

Additiv homomorphe Verschlüsselungsverfahren erlauben die Addition unter Verschlüsselung, d.h. für alle möglichen Klartexte x, y aus dem Klartextraum P gilt:

$$\forall x, y \in P : [\![x]\!] \boxplus [\![y]\!] = [\![x + y]\!].$$

Wiederholtes Verdoppeln und Addieren erlaubt Multiplikation mit einer Konstanten a:

$$\forall a \in \mathbb{Z}, x \in P : a[\![x]\!] = [\![ax]\!].$$

Das am weitesten verbreitete additiv homomorphe Verschlüsselungsverfahren ist das von Paillier [Pai99] mit Optimierungen aus [DJ01]. Der öffentliche Schlüssel dieses Verfahrens ist ein RSA Modul n, d.h. das Produkt zweier sehr großer[1] Primzahlen p, q, und der geheime Schlüssel die Faktorisierung p, q von n; der Klartextraum P ist $\mathbb{Z}_n = [0, \ldots, n-1]$.

Multiplikation unter additiv homomorpher Verschlüsselung erfordert eine Runde Interaktion zwischen $\mathcal{S}$ und $\mathcal{C}$: Hierzu addiert $\mathcal{S}$ zu den zu multiplizierenden Chiffretexten $[\![x]\!]$ und $[\![y]\!]$ eine zufällig gewählte Maske r_x bzw. r_y und schickt $[\![\bar{x}]\!] = [\![x]\!] \boxplus [\![r_x]\!]$, $[\![\bar{y}]\!] = [\![y]\!] \boxplus [\![r_y]\!]$ an $\mathcal{C}$. $\mathcal{C}$ entschlüsselt, multipliziert und schickt $[\![\bar{x} \cdot \bar{y}]\!]$ zurück an $\mathcal{S}$. Schließlich berechnet $\mathcal{S}$ das gewünschte homomorph verschlüsselte Produkt als

$$[\![x \cdot y]\!] = [\![\bar{x} \cdot \bar{y}]\!] \boxplus (-r_x)[\![y]\!] \boxplus (-r_y)[\![x]\!] \boxplus [\![-r_x r_y]\!] = [\![(x+r_x)(y+r_y) - r_x y - r_y x - r_x r_y]\!].$$

Voll homomorphe Verschlüsselungsverfahren erlauben neben der Addition auch die (nicht-interaktive) Multiplikation unter Verschlüsselung. Da Addition und Multiplikation eine vollständige Boole'sche Basis bilden erlaubt ein solches Verschlüsselungsverfahren die Berechnung *beliebiger* Funktionen unter Verschlüsselung.

Lange Zeit waren lediglich voll homomorphe Verfahren mit begrenzter Anzahl an Multiplikationen bekannt, z.B. maximal eine Multiplikation [BGN05, GHV10] oder Chiffretexte, die exponentiell in der Anzahl der Multiplikationen wachsen [SYY99]. Die kürzlich vorgestellten voll homomorphen Verschlüsselungsverfahren [Gen09, SV10, DGHV10] unterliegen zwar theoretisch nicht mehr solchen Einschränkungen, sind jedoch für praktische Anwendungen noch viel zu ineffizient.

Zur Zeit sind die in der Praxis effizientesten Verfahren zum Rechnen auf verschlüsselten Daten somit die additiv homomorphen Verfahren mit interaktiver Multiplikation.

2.2 Rechnen mit verschlüsselten Funktionen

Beim Rechnen mit verschlüsselten Funktionen wird eine verschlüsselte Funktion (engl. "garbled functions") auf den verschlüsselten Daten berechnet. Hierbei wird die Funktion nicht auf Klartextwerten ('0' oder '1'), sondern auf zufällig gewählten symmetrischen Schlüsseln (engl. "garbled values") berechnet [Yao86]. Im Vergleich zum Rechnen auf verschlüsselten Daten mit homomorpher Verschlüsselung (vgl. §2.1) kann auf diesen symmetrischen Schlüsseln nicht direkt gerechnet werden, sondern nur unter Verwendung von verschlüsselten Übersetzungstabellen (engl. "garbled tables") für jede Elementaroperation, die in der Setup Phase übertragen werden. Die anschließende Online Phase

[1] Nach aktuellen Empfehlungen sollten die beiden Faktoren etwa tausend Bit lang sein [GQ09].

von Protokollen zum Rechnen mit verschlüsselten Funktionen ist sehr effizient, da lediglich kryptographische Hashfunktionen berechnet werden müssen und keine rechenaufwändigen public-key Operationen wie beim Rechen auf homomorph verschlüsselten Daten [KSS09, KSS10].

2.2.1 Yao's Protokoll

Zusammengefasst funktioniert Yao's Protokoll zum Rechnen mit verschlüsselten Schaltkreisen [Yao86] wie in Abbildung 1 gezeigt:

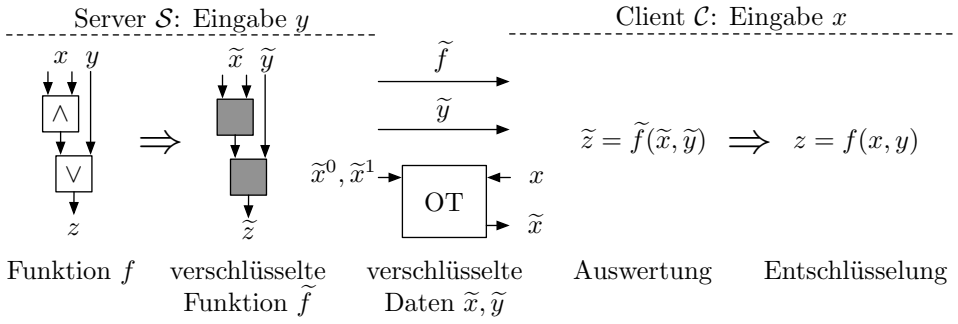

Abbildung 1: Yao's Protokoll zum Rechnen mit verschlüsselten Funktionen

In der Setup Phase generiert der Server $\mathcal{S}$ die verschlüsselte Funktion $\widetilde{f}$ aus der als Boole'scher Schaltkreis repräsentierten zu berechnenden Funktion f. Hierfür weist $\mathcal{S}$ jeder Kante von f zwei zufällige symmetrische Schlüssel zu $\widetilde{w}_i^0, \widetilde{w}_i^1$, die den entsprechenden Klartextwerten 0 bzw. 1 entsprechen. Da beide Schlüssel zufällig sind liefert $\widetilde{w}_i^j$ keine Information über den zugehörigen Klartextwert j. Nun berechnet $\mathcal{S}$ für jedes Gatter von f eine verschlüsselte Funktionstabelle, die es erlaubt, bei Kenntnis der Schlüssel für die Eingabekanten dieses Gatters, den entsprechenden Schlüssel für die Ausgabekante des Gatters zu berechnen. Die verschlüsselte Funktion $\widetilde{f}$ besteht aus den verschlüsselten Funktionstabellen für jedes Gatter und wird an $\mathcal{C}$ geschickt.

Später, in der Online Phase des Protokolls, erhält $\mathcal{C}$ die zu den Eingaben x und y gehörenden Schlüssel $\widetilde{x}$ und $\widetilde{y}$ und kann die verschlüsselte Funktion $\widetilde{f}$ Gatter für Gatter auswerten und erhält die verschlüsselte Ausgabe $\widetilde{z} = \widetilde{f}(\widetilde{x}, \widetilde{y})$, die mit Hilfe von $\mathcal{S}$ in die zugehörige Klartextausgabe $z = f(x, y)$ entschlüsselt werden kann.

Um ein Bit y_i der Eingabe von $\mathcal{S}$ zu verschlüsseln, sendet $\mathcal{S}$ einfach den zugehörigen Schlüssel $\widetilde{y}_i = \widetilde{y}_i^{y_i}$ an $\mathcal{C}$. Analog soll $\mathcal{C}$ den verschlüsselten Wert $\widetilde{x}_i = \widetilde{x}_i^{x_i}$ erhalten jedoch so, dass $\mathcal{S}$ nicht den Klartextwert x_i erfährt. Dies wird mit einem kryptographischen Protokoll genannt Oblivious Transfer (OT) erreicht (Details in §A).

Die verschlüsselte Funktion darf nur *genau einmal ausgewertet* werden, da ansonsten $\mathcal{C}$ aus zuvor gesehenen Schlüsseln Informationen über die Eingabewerte von $\mathcal{S}$ lernen kann. Für den Sicherheitsbeweis des skizzierten Protokolls von Yao verweisen wir auf [LP09], für effiziente Instanzierungen auf [PSSW09] und weitere Details auf [KSS10].

2.3 Kombiniertes Rechnen unter Verschlüsselung

Das in [KSS10] vorgeschlagene Modell erlaubt es, Protokolle zum Rechnen unter Verschlüsselung als Sequenz von Operationen auf verschlüsselten Daten zu beschreiben wie in Abbildung 2 gezeigt: Beide Parteien haben zunächst Klartext Werte x als Eingaben. Diese werden zunächst in verschlüsselte Werte übersetzt, auf denen verschlüsselt gerechnet werden kann. Abschließend wird das verschlüsselte Endergebnis wieder zurück in einen Klartext Wert für Client $\mathcal{C}$ oder Server $\mathcal{S}$ umgewandelt.

Je nachdem, welche Operationen unter Verschlüsselung berechnet werden sollen stehen zwei verschiedene Arten der Verschlüsselung zur Verfügung: "Homomorphe Werte" $[\![x]\!]$ zum Rechnen unter homomorpher Verschlüsselung (vgl. §2.1) oder "Garbled Werte" $\widetilde{x}$ zum Rechnen mit verschlüsselten Funktionen (vgl. §2.2), die auch ineinander umgewandelt werden können ($\widetilde{x} \Leftrightarrow [\![x]\!]$).

Die Kombination dieser beiden Verfahren erlaubt es, das jeweils effizienteste Verfahren für eine bestimmte Teilfunktionalität zu verwenden. So erlaubt homomorphe Verschlüsselung beispielsweise effiziente (interaktive) Multiplikation für große Werte [HKS+10], während verschlüsselte Schaltkreise für Vergleichsoperationen besser geeignet sind [KSS09].

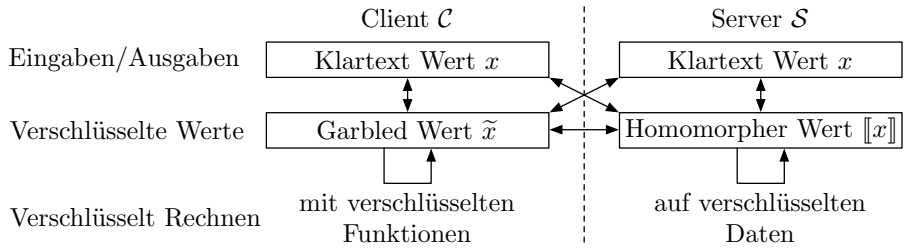

Abbildung 2: Modell zum kombinierten Rechnen unter Verschlüsselung

3 TASTY: Werkzeug zum Rechnen unter Verschlüsselung

Im Folgenden stellen wir TASTY (Tool for Automating Secure Two-partY computations) [HKS+10] vor, ein Werkzeug, das es erlaubt, Protokolle zum Rechnen unter Verschlüsselung zu beschreiben und automatisch zu generieren.

Existierende Ansätze. Im Gegensatz zu vorherigen Werkzeugen wie Fairplay [MNPS04, BDNP08] oder VIFF [DGKN09], die jeweils auf ein Paradigma zum Rechnen unter Verschlüsselung beschränkt waren (vgl. §2.1 bzw. §2.2), unterstützt TASTY die Kombination beider Verfahren, d.h. homomorphe Verschlüsselung und verschlüsselte Schaltkreise (vgl. §2.3), um hoch effiziente Protokolle zu erhalten.

TASTY verfolgt unter anderem die folgenden Design-Ziele:

1. Protokolle zum kombinierten Rechnen unter Verschlüsselung können in TASTYL programmiert werden, einer intuitiven Hochsprache, die solche Protokolle als Sequenz von Operationen auf verschlüsselten Daten beschreibt (vgl. §3.2).
2. Da besonders in Web-Applikationen die Antwortzeit kritisch ist optimiert TASTY die generierten Protokolle dahingehend, dass die Latenz der Online Phase, d.h. die Zeit von der Eingabe der Daten bis zur Ausgabe des Ergebnisses, minimiert wird. Dies wird dadurch erreicht, dass rechenaufwändige kryptographische Operationen und ein Großteil der Daten bereits in einer Setup Phase vorberechnet und ausgetauscht werden.

3.1 TASTY: Architektur und Workflow

Der Workflow zur Benutzung von TASTY ist wie in Abbildung 3 gezeigt:

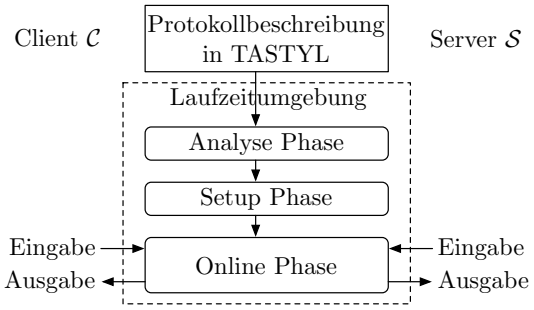

Abbildung 3: Architektur und Workflow von TASTY

1. Die beiden Benutzer, Client $\mathcal{C}$ und Server $\mathcal{S}$, einigen sich auf eine *Protokollbeschreibung* des zu berechnenden Protokolls in der TASTY Eingabesprache (TASTYL), die in §3.2 beschrieben wird.
2. Beide Benutzer rufen die *Laufzeitumgebung* von TASTY auf, ein Programm, welches das Protokoll analysiert und ausführt:
   (a) In der *Analyse Phase* überprüft die Laufzeitumgebung die syntaktische Korrektheit der Protokollbeschreibung, stellt sicher, dass beide Parteien das selbe Protokoll ausführen und ermittelt automatisch, welche Teile des Protokolls vorberechnet werden können.
   (b) In der *Setup Phase* führen beide Parteien die Vorberechnungen aus.
   (c) In der *Online Phase* geben beide Parteien ihre Eingaben ein und das Protokoll berechnet die jeweiligen Ausgaben unter Verschlüsselung.

3.2 TASTYL: Sprache zum kombinierten Rechnen unter Verschlüsselung

TASTYL, die "TASTY input Language", ist eine domänenspezifische Hochsprache zum Beschreiben von Protokollen zum kombinierten Rechnen unter Verschlüsselung als Sequenz von Operationen auf verschlüsselten Daten.

Im Folgenden fassen wir einige Grundelemente des Typsystems von TASTYL wie in Abbildung 4 zusammen. Ein Beispielprogramm in TASTYL ist in §B aufgelistet. Für Details zu TASTY und weitere Programmbeispiele verweisen wir auf [HKS+10].

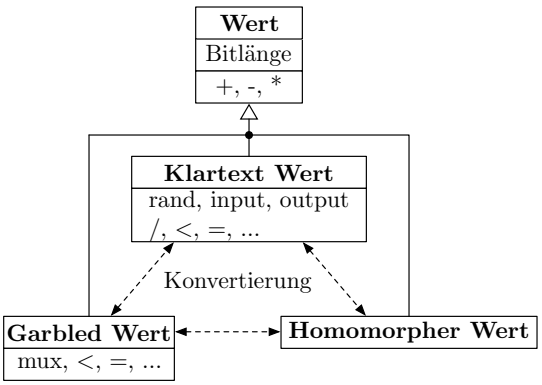

Abbildung 4: Typsystem von TASTYL (vereinfacht)

Jede Variable in TASTYL ist ein *Wert* (s. Abbildung 4) mit einer festen Bitlänge oder ein mehrdimensionaler Vektor bestehend aus mehreren gleichartigen Werten, auf die wir im Folgenden nicht näher eingehen werden. Jeder Wert kann entweder ein unverschlüsselter *Klartext Wert* oder ein verschlüsselter *Garbled Wert* bzw. *Homomorphic Wert* sein. Alle Werte unterstützen die Operationen Addition, Subtraktion und Multiplikation.

Zusätzlich zu diesen gemeinsam Operatoren werden weitere Operatoren und Konvertierungen angeboten:

Klartext Werte. Eingaben (`input`) und Ausgaben (`output`) der beiden Parteien sind Klartext Werte. Diese können zufällig gewählt werden (`rand`) und bieten zuätzliche Operationen wie Integer Division und Vergleiche.

Homomorphe Werte. Klartext Werte können in homomorph verschlüsselte *Homomorphe Werte* konvertiert werden und umgekehrt. Homomorphe Werte können nichtinteraktiv addiert, subtrahiert und mit einem Klartext Wert auf Seite von $\mathcal{S}$ multipliziert werden – die Multiplikation zweier Homomorpher Werte erfordert eine Runde Interaktion (vgl. §2.1).

Garbled Werte. Sowohl Klartext Werte als auch Homomorphe Werte können in verschlüsselte *Garbled Werte* konvertiert werden und umgekehrt. Der Vergleich zweier Garbled Werte liefert ein Garbled Bit (Garbled Wert mit Bitlänge 1), welches wiederum verwendet werden kann, um zwischen zwei Garbled Werten auszuwählen (Multiplexer `mux`). Für jede Operation auf Garbled Werten erzeugt TASTY dynamisch den entsprechenden verschlüsselten Schaltkreis.

4 Szenario: Klassifikation von EKG Daten unter Verschlüsselung

Als medizinisches Anwendungsszenario betrachten wir einen Web-Service, der biometrische Daten unter Verschlüsselung klassifizieren soll. Aus dem breiten Spektrum verschiedener biomedizinischer Daten [BCA+93, TWBT95, FLK08, KTB+03] wählen wir Elektrokardiogramm (EKG) Daten als einfaches aber repräsentatives Beispiel.

Ein Patient (Client $\mathcal{C}$) hat ein EKG erstellt und möchte dieses bei einem Service Anbieter (Server $\mathcal{S}$) klassifizieren lassen. Bezüglich der Privatsphäre ergeben sich folgende Anforderungen: $\mathcal{C}$ wünscht, dass $\mathcal{S}$ keinerlei Information über das EKG Signal erhält, da dies sensitive personenbezogene Daten sind). Andererseits möchte $\mathcal{S}$ Details über den verwendeten Klassifikationsalgorithmus vor $\mathcal{C}$ verbergen, da dies wertvolles geistiges Eigentum von $\mathcal{S}$ darstellt. Im Folgenden nehmen wir an, dass die Struktur des verwendeten Klassifikationsalgorithmus beiden Parteien bekannt ist und das schützenswerte geistige Eigentum von $\mathcal{S}$ die Parametrisierung des Algorithmus ist.

Eine solche sichere Klassifikation von EKG Daten kann dadurch erreicht werden, dass ein Algorithmus zur Klassifikation von EKG Daten unter Verschlüsselung berechnet wird: Hierfür wird zunächst ein bestehender Algorithmus zur EKG Klassifikation von Klartextdaten gewählt (in unserem Beispiel [ASSK07, GSK02]) und auf Integer-Arithmetik abgebildet (im Beispiel wie in [BFK+09a, BFK+09b] beschrieben). Anschließend kann dieser Algorithmus in TASTYL programmiert werden (s. §B). Schließlich wird dieses TASTYL Programm sicher von der TASTY Laufzeitumgebung berechnet.

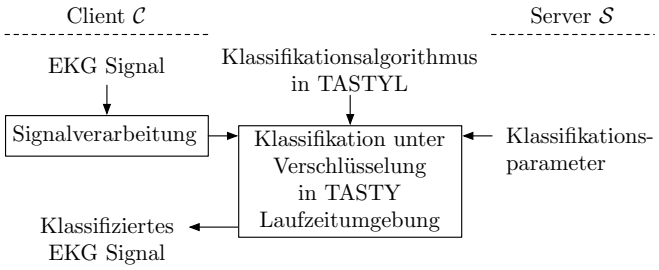

Abbildung 5: EKG Klassifikation unter Verschlüsselung mit TASTY

Der Gesamtablauf ist in Abbildung 5 gezeigt: Das EKG Signal von $\mathcal{C}$ wird zunächst vorverarbeitet (Rauschfilterung, Quantisierung, etc. – Details s. [BFK+09a, BFK+09b]). Der in TASTYL beschriebene Klassifikationsalgorithmus (s. §B) bekommt das verarbei-

tete EKG Signal als (geheime) Eingabe von $\mathcal{C}$ und die Parameter zur Konfiguration des Klassifikationsalgorithmus als (geheime) Eingabe von $\mathcal{S}$. Die TASTY Laufzeitumgebung berechnet die Klassifikation unter Verschlüsselung und gibt das klassifizierte EKG Signal an $\mathcal{C}$ aus.

5 Diskussion des Ansatzes

Abschließend möchten wir den Ansatz des Rechnens unter Verschüsselung und seine Verwendbarkeit insbesondere im Hinblick auf medizinische Anwendungen kurz diskutieren.

Effizienz. Protokolle zum Rechnen unter Verschlüsselung sind naturgemäß langsamer als wenn die Berechnungen unverschlüsselt durchgeführt werden. Als Faustregel für die Effizienz gilt, dass Protokolle zum Rechnen mit verschlüsselten Funktionen (s. §2.2) in der Online Phase in etwa eine symmetrische Entschlüsselung pro Bit-Operation kosten, während Berechnungen unter additiv homomorpher Verschlüsselung (s. §2.1) langsamer als bestehende public-key Verfahren wie RSA sind.

Dank der rasanten Zunahme verfügbarer Rechenleistung und rapide steigender Übertragungsraten von Datennetzen, in Kombination mit mehreren algorithmischen Verbesserungen der zugrunde liegenden Primitiven und Protokolle, ist Rechnen unter Verschlüsselung jedoch in den Bereich der Anwendbarkeit in der Praxis gerückt, wie beispielsweise ein System zur sicheren Gesichtserkennung zeigt [OPJM10].

Die sichere Klassifikation von EKG Daten auf aktueller PC Hardware benötigt ca. 19 s Rechenzeit und 64 kByte Datentransfer [BFK+09a].

Sicherheit. Der beim Rechnen unter Verschlüsselung verwendete Sicherheitsparameter bestimmt die Zeitspanne, bis zu der die Verschlüsselung als sicher angenommen werden kann. Die meisten existierenden prototypischen Implementierungen zum Rechnen unter Verschlüsselung basieren jedoch auf einem symmetrischen Sicherheitsparameter von 80 bit, dessen Verwendung nur noch bis Ende 2010 empfohlen wird [GQ09].

Insbesondere in medizinischen Anwendungen, in denen sensible Patientendaten unter Verschlüsselung verarbeitet werden, ist es notwendig, dass mitgeschnittene Protokollläufe auch in Zukunft nicht entschlüsselt werden können. Hierfür sollte ein ausreichend großer Sicherheitsparameter gewählt werden (z.B: 128 bit, dessen Sicherheit bis zum Jahr 2040 prognostiziert wird [GQ09]). Die Verwendung eines größeren Sicherheitsparameters hat jedoch negative Auswirkungen auf die Performanz der Protokolle. Beispielsweise dauert die Klassifikation von EKG Daten mit 112 bit Sicherheitsparameter (empfohlen bis zum Jahr 2030) ca. 41 s und 89 kByte.

Protokolle zum Rechnen unter Verschlüsselung mit *informationstheoretischer Sicherheit*, d.h. Protokolle, die mit unendlicher Rechenleistung nicht zu brechen sind, sind zwar bekannt [BOGW88, CCD88], jedoch zu komplex für den Einsatz in der Praxis.

In manchen Anwendungen ist die in vielen Papieren getroffene Annahme, dass die betei-

ligten Parteien ehrlich aber neugierig (engl. “semi-honest adversaries”) sind, also nicht in bösartiger Absicht vom Protokoll abweichen, ebenfalls nicht gerechtfertigt. Bestehende Protokolle zum Rechnen unter Verschlüsselung, die solche Betrugsversuche mit hoher (engl. “covert adversaries”) oder an Sicherheit grenzender (engl. “malicious adversaries”) Wahrscheinlichkeit erkennen sind jedoch deutlich aufwändiger. Beispielsweise benötigt die sichere Berechnung der aus 33.880 Bit-Operationen bestehenden AES Funktionalität mit Sicherheitsparameter 128 bit im semi-honest Fall 7 s ($0,5$ MByte), im covert Fall 60 s ($8,7$ MByte) und im malicious Fall 19 min (408 MByte) [PSSW09].

Benutzbarkeit. Neben der Performanz und Sicherheit der Protokolle zum Rechnen unter Verschlüsselung ist ihre Benutzbarkeit der wohl wichtigste Aspekt für den Einsatz in praktischen Anwendungen, um Entwicklungskosten zu minimieren.

Für den Entwurf effizienter und sicherer Protokolle war lange Zeit ein detailliertes Expertenwissen nötig. Auch bei der Implementierung der Protokolle konnten sich diverse Programmierfehler einschleichen, die die Sicherheit gefährdeten. Dies ist vergleichbar mit Zeiten, in denen ausschließlich in Maschinensprache (Assembler) programmiert wurde.

Compiler für kryptographische Protokolle (wie das in §3 beschriebene Werkzeug TASTY) abstrahieren von den kryptographischen Details und erlauben es so Anwendungsentwicklern auch ohne kryptographischem Hintergrundwissen die zu berechnende Funktionalität in einer Hochsprache zu beschreiben. Aus dieser Hochsprachenbeschreibung wird dann automatisch ein ausführbares, effizientes und sicheres Protokoll erzeugt. In Analogie abstrahieren Hochsprachen wie Java, C oder C++ von maschinenspezifischen Details und generieren aus Programmen effizienten und korrekten (im Sinne von äquivalent zur Hochsprachenbeschreibung) Maschinencode und erhöhen somit die Produktivität.

Im EU Projekt CACE (Computer Aided Cryptography Engineering)[2] werden neben TASTY eine Vielzahl von Werkzeugen zur automatischen Generierung von kryptographischen Protokollen und Primitiven entwickelt [BBB+10].

Nutzerfreundlichkeit. Zusammengefasst ist das Rechnen unter Verschlüsselung bereits heute technisch möglich und kann prinzipiell als Grundlage für vielerlei innovative und Datenschutz-konforme Anwendungen genutzt werden.

Um eine hohe Akzeptanz solcher Systeme zu erreichen ist neben dem Aspekt der Sicherheit jedoch vor allem die Nutzerfreundlichkeit von herausragender Bedeutung. Damit die Patienten solche neuartigen Dienste akzeptieren und als Ergänzung zum persönlichen Arztbesuch ansehen, ist vor allem eine einfach zu bedienende Benutzerschnittstelle wichtig, die sich nahtlos in den gewohnten Alltag der Patienten integriert.

[2] `http://www.cace-project.eu`

Danksagung

Diese Arbeit wurde gefördert durch das NRW Projekt MediTrust und das EU Projekt CACE (Computer Aided Cryptography Engineering).

Literatur

[ASSK07] U. R. Acharya, J. Suri, J. A. E. Spaan und S. M. Krishnan. *Advances in Cardiac Signal Processing*, Kapitel 8. Springer, 2007.

[BBB+10] E. Bangerter, M. Barbosa, D.J. Bernstein, I. Damgard, D. Page, J.I. Pagter, A.-R. Sadeghi und S. Sovio. Using Compilers to Enhance Cryptographic Product Development. In *Information Security Solutions Europe (ISSE'10)*, Seiten 291–301. Vieweg+Teubner, 2010.

[BCA+93] F. M. Bennett, D. J. Christini, H. Ahmed, K. Lutchen, J. M. Hausdorff und N. Oriol. Time series modeling of heart rate dynamics. In *Computers in Cardiology 1993. Proceedings.*, Seiten 273–276, 1993.

[BDNP08] A. Ben-David, N. Nisan und B. Pinkas. FairplayMP: a system for secure multi-party computation. In *ACM Conference on Computer and Communications Security (CCS'08)*, Seiten 257–266. ACM, 2008. `http://fairplayproject.net/fairplayMP.html`.

[Bea95] D. Beaver. Precomputing Oblivious Transfer. In *Advances in Cryptology – CRYPTO'95*, *LNCS*, Band 963, Seiten 97–109. Springer, 1995.

[BFK+09a] M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A. Paus, A.-R. Sadeghi und T. Schneider. Efficient Privacy-Preserving Classification of ECG Signals. In *IEEE International Workshop on Information Forensics and Security (IEEE WIFS'09)*, Seiten 91–95. IEEE, 2009.

[BFK+09b] M. Barni, P. Failla, V. Kolesnikov, R. Lazzeretti, A.-R. Sadeghi und T. Schneider. Secure Evaluation of Private Linear Branching Programs with Medical Applications. In *European Symposium on Research in Computer Security (ESORICS'09)*, *LNCS*, Band 5789, Seiten 424–439. Springer, 2009.

[BGN05] D. Boneh, E.-J. Goh und K. Nissim. Evaluating 2-DNF Formulas on Ciphertexts. In *Theory of Cryptography Cronference (TCC'05)*, *LNCS*, Band 3378, Seiten 325–341. Springer, 2005.

[Bla08] D. Blankenhorn. Microsoft HealthVault is nothing like Google Health, 2008.

[BOGW88] M. Ben-Or, S. Goldwasser und A. Wigderson. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *Symposium on Theory of Computing (STOC'88)*, Seiten 1–10. ACM, 1988.

[CCD88] D. Chaum, C. Crépeau und I. Damgård. Multiparty Unconditionally Secure Protocols (Extended Abstract). In *Symposium on Theory of Computing (STOC'88)*, Seiten 11–19. ACM, 1988.

[DGHV10] M. v. Dijk, C. Gentry, S. Halevi und V. Vaikuntanathan. Fully Homomorphic Encryption over the Integers. In *Advances in Cryptology – EUROCRYPT'10*, *LNCS*, Band 6110, Seiten 24–43. Springer, 2010.

[DGKN09] I. Damgård, M. Geisler, M. Krøigård und J. B. Nielsen. Asynchronous Multiparty Computation: Theory and Implementation. In *Public Key Cryptography (PKC'09)*, *LNCS*, Band 5443, Seiten 160–179. Springer, 2009. `http://viff.dk`.

[DJ01] I. Damgård und M. Jurik. A Generalisation, a Simplification and some Applications of Paillier's Probabilistic Public-Key System. In *Public-Key Cryptography (PKC'01)*, *LNCS*, Band 1992, Seiten 119–136. Springer, 2001.

[FLK08] P. Flor-Henry, J. L. Lind und Z. J. Koles. Quantitative EEG and source localization in fibromyalgia. *International Journal of Psychophysiology*, 69(3):142–142, 2008.

[Gen09] C. Gentry. Fully homomorphic encryption using ideal lattices. In *ACM Symposium on Theory of Computing (STOC'09)*, Seiten 169–178. ACM, 2009.

[GHV10] C. Gentry, S. Halevi und V. Vaikuntanathan. A Simple BGN-type Cryptosystem from LWE. In *Advances in Cryptology – EUROCRYPT'10*, *LNCS*, Band 6110, Seiten 506–522. Springer, 2010.

[GQ09] D. Giry und J.-J. Quisquater. Cryptographic Key Length Recommendation, March 2009. `http://keylength.com`.

[GSK02] D. F. Ge, N. Srinivasan und S. M. Krishnan. Cardiac arrhythmia classification using autoregressive modeling. *BioMedical Engineering OnLine*, 1(1):5, 2002.

[HKS+10] W. Henecka, S. Kögl, A.-R. Sadeghi, T. Schneider und I. Wehrenberg. TASTY: Tool for Automating Secure Two-partY computations. In *ACM Conference on Computer and Communications Security (CCS'10)*. ACM, October 4-8, 2010. `http://tastyproject.net`.

[IKNP03] Y. Ishai, J. Kilian, K. Nissim und E. Petrank. Extending Oblivious Transfers Efficiently. In *Advances in Cryptology – CRYPTO'03*, *LNCS*, Band 2729, Seiten 145–161. Springer, 2003.

[KSS09] V. Kolesnikov, A.-R. Sadeghi und T. Schneider. Improved Garbled Circuit Building Blocks and Applications to Auctions and Computing Minima. In *Cryptology and Network Security (CANS'09)*, *LNCS*, Band 5888, Seiten 1–20. Springer, 2009.

[KSS10] V. Kolesnikov, A.-R. Sadeghi und T. Schneider. Modular Design of Efficient Secure Function Evaluation Protocols. Cryptology ePrint Archive, Report 2010/079, 2010. `http://eprint.iacr.org/2010/079/`.

[KTB+03] P. Kalra, J. Togami, G. Bansal, A. W. Partin, M. K. Brawer, R. J. Babaian, L. S. Ross und C. S. Niederberger. A neurocomputational model for prostate carcinoma detection. *Cancer*, 98(9):1849–1854, 2003.

[LP09] Y. Lindell und B. Pinkas. A Proof of Yao's Protocol for Secure Two-Party Computation. *Journal of Cryptology*, 22(2):161–188, 2009. Cryptology ePrint Archive: Report 2004/175.

[McB08] M. McBride. Google Health: Birth of a giant. *Health Management Technology*, 29:8–10, 2008.

[MNPS04] D. Malkhi, N. Nisan, B. Pinkas und Y. Sella. Fairplay — a secure two-party computation system. In *USENIX Security Symposium*, 2004. `http://fairplayproject.net/fairplay.html`.

[NP01] M. Naor und B. Pinkas. Efficient oblivious transfer protocols. In *ACM-SIAM Symposium On Discrete Algorithms (SODA'01)*, Seiten 448–457. Society for Industrial and Applied Mathematics, 2001.

[OPJM10] M. Osadchy, B. Pinkas, A. Jarrous und B. Moskovich. SCiFI - A System for Secure Face Identification. In *IEEE Symposium on Security & Privacy (S&P'10)*, Seiten 239–254. IEEE, 2010.

[Pai99] P. Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology – EUROCRYPT'99*, *LNCS*, Band 1592, Seiten 223–238. Springer, 1999.

[PSSW09] B. Pinkas, T. Schneider, N. P. Smart und S. C. Williams. Secure Two-Party Computation is Practical. In *Advances in Cryptology – ASIACRYPT'09*, *LNCS*, Band 5912, Seiten 250–267. Springer, 2009.

[SV10] N. P. Smart und F. Vercauteren. Fully Homomorphic Encryption with Relatively Small Key and Ciphertext Sizes. In *Public Key Cryptography (PKC'10)*, *LNCS*, Band 6056, Seiten 420–443. Springer, 2010.

[SYY99] T. Sander, A. Young und M. Yung. Non-Interactive CryptoComputing for NC^1. In *IEEE Symposium on Foundations of Computer Science (FOCS'99)*, Seiten 554–566. IEEE, 1999.

[TWBT95] S. Todd, M. Woodward, C. Bolton-Smith und H. Tunstall-Pedoe. An investigation of the relationship between antioxidant vitamin intake and coronary heart disease in men and women using discriminant analysis. *Journal of Clinical Epidemiology*, 48(2):297–305, 1995.

[Yao86] A. C. Yao. How to Generate and Exchange Secrets. In *IEEE Symposium on Foundations of Computer Science (FOCS'86)*, Seiten 162–167. IEEE, 1986.

A Oblivious Transfer

Oblivious Transfer (OT) ist ein Protokoll, dessen Eingaben ein Bit b von $\mathcal{C}$ und zwei Nachrichten s^0 und s^1 von $\mathcal{S}$ sind. OT garantiert, dass $\mathcal{C}$ ausschließlich die gewählte Nachricht s^b erhält und keinerlei Information über s^{1-b}, während $\mathcal{S}$ die Wahl b nicht lernt.

Im Kontext des in §2.2.1 beschriebenen Protokolls von Yao wird für jedes Eingabebit x_i von $\mathcal{C}$ ein OT Protokoll ausgeführt, mit dem $\mathcal{C}$ den zugehörigen Schlüssel $\widetilde{x}_i$ erhält, ohne dass $\mathcal{S}$ den Wert x_i lernt: Die Eingabe von $\mathcal{C}$ ist hierbei das Auswahlbit $b = x_i$ und die Eingabe von $\mathcal{S}$ sind die beiden zugehörigen Schlüssel $s^0 = \widetilde{x}_i^0$ und $s^1 = \widetilde{x}_i^1$. Als Ausgabe erhält $\mathcal{C}$ den Wert $s^b = \widetilde{x}_i^{x_i} = \widetilde{x}_i$.

Durch Kombination verschiedener Optimierungen kann OT sehr günstig implementiert werden (für Details s. [KSS10]): Die Konstruktion von Beaver [Bea95] erlaubt es, alle OTs in der Setup Phase vor zu berechnen, so dass die Online Phase im Wesentlichen aus dem Versenden zweier Nachrichten besteht. Mit der Konstruktion von Ishai et al. [IKNP03] kann die Komplexität der Setup Phase weiter verringert werden, indem eine beliebige Anzahl paralleler OTs auf eine konstante Anzahl von OTs reduziert wird. Die verbleibende konstante Anzahl OTs in der Setup Phase kann mit effizienten OT Protokollen über elliptischen Kurven implementiert werden, z.B. [NP01].

B EKG Klassifikation in TASTYL

```
def protocol(c, s):
    L = 24
    N = 15
    D = 6

    # Eingaben
    c.x = SignedVec(bitlen=L, dim=N)
    s.A = SignedVec(bitlen=L, dim=(D, N))

    # Berechne y = Ax auf verschluesselten Daten
    s.hx <<= HomomorphicVec(val=c.x)
    s.hy = HomomorphicVec(bitlen=L, dim=D)
    for i in xrange(D):
        s.hy[i] = s.A[i].dot(s.hx)

    # Berechne y[i] > 0 mit verschluesselter Funktion
    c.gy <<= GarbledVec(val=s.hy)
    c.gs = GarbledVec(bitlen=1, dim=D)
    for i in xrange(D):
        c.gs[i] = c.gy[i] > 0

    # Berechne Entscheidungsdiagramm
    c.VF = Unsigned(val=(not c.gs[0]) and (not c.gs[2]))
    c.VT = Unsigned(val=(not c.gs[0]) and c.gs[2])
    c.SVT = Unsigned(val=c.gs[0] and (not c.gs[2]))
    c.gt = c.gs[0] and c.gs[1]
    c.PVC = Unsigned(val=c.gt and (not c.gs[3]) and (not c.gs[4]))
    c.APC = Unsigned(val=c.gt and c.gs[3] and (not c.gs[5]))

    # Ausgabe
    if c.VF:
        c.output("Ventricular Fibrillation (VF)")
    elif c.VT:
        c.output("Ventricular Tachycardia (VT)")
    elif c.SVT:
        c.output("SupraVentricular Tachycardia (SVT)")
    elif c.PVC:
        c.output("Premature Ventricular Contraction (PVC)")
    elif c.APC:
        c.output("Atrial Premature Contraction (APC)")
    else:
        c.output("Normal Sinus Rhythm (NSR)")
```