Private Policy Negotiation

Klaus Kursawe [*] Gregory Neven [†] Pim Tuyls [‡]

April 14, 2006

Abstract

With the increasing importance of correctly handling privacy-sensitive data, significant work has been put in expressing and enforcing privacy policies. Less work has been done however on *negotiating* a privacy policy, especially if the negctiation process itself is considered privacy-sensitive. In this paper, we present a formal definition of the *mutually privacy-preserving policy negotiation problem*, i.e. the problem of negotiating what data will be revealed under what conditions, while no party learns anything about the other parties' preferences other than the outcome of the negotiation.

We validate the definition by providing a reference solution using two-party computation techniques based on homomorphic encryption systems. Based on an evaluation of the efficiency of our protocol in terms of computation, bandwidth and communication rounds, we conclude that our solution is practically feasible for simple policies or high-bandwidth communication channels.

Keywords: Policy negotiation, privacy, anonymity, two-party computation.

[*]Dept. of Electrical Engineering, Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, 3001 Heverlee-Leuven, Belgium. E-Mail: `Klaus.Kursawe@esat.kuleuven.ac.be`. URL: `http://www.esat.kuleuven.ac.be/~kkursawe`.

[†]Dept. of Electrical Engineering, Katholieke Universiteit Leuven, Kasteelpark Arenberg 10, 3001 Heverlee-Leuven, Belgium; and Département d'Informatique, Ecole Normale Supérieure, 45 rue d'Ulm, 75230 Paris Cedex 05, France. E-Mail: `Gregory.Neven@esat.kuleuven.ac.be`. URL: `http://www.neven.org`.

[‡]Philps Research, Professor Holstlaan 4, 5656 AA Eindhoven, The Netherlands. E-Mail: `Pim.Tuyls@philips.com`.

Contents

1. Introduction

With the increasing amount of electronic data produced by day-to-day interactions, as well as the ability to link or otherwise process this data, the handling of privacy-sensitive personal data has emerged as an important field in computer security. Many online services require the user to submit some information about himself (e.g. name, address, ...) in order to access the service. The type of information to be provided is described in a *policy*.

Substantial work has been done on defining privacy policies (e.g. P3P [W3C02] and EPAL [BKBS04]), and their enforcement [MPB03, BM05b, CV02]. Less work however has been done on policy *negotiation*. Usually, it is assumed that both sides somehow agree on a common policy specifying what data will be transmitted and how sensitive data should be handled. In various settings, this negotiation is complicated by the effect that a person's privacy preferences may already give away information about that person. Since the vast majority of the population is still willing to reveal seemingly innocent data (e.g. their consumption of alcohol), a person that considers this piece of data as sensitive might quickly raise suspicion and consequently be treated with a worst-case assumption (e.g. that he's an alcoholic). One can argue that in the negotiation between a consumer and a Company or Government Entity, the latter one has no right for privacy, and that transparency about the preferences is expected. However, in a peer-to-peer scenario, e.g. between two consumers or two companies, it is important to protect both sides.

Given that both parties' preferences themselves are to be considered as private data, can they still discover whether a matching policy exists, and if so, what this policy is? In this paper, we answer this question in a positive way.

1.1. Our Contributions

We formally define the problem of negotiating a privacy policy from two sets of preferences, in such a way that no party learns any information about the other's preferences other than the policy agreed upon. We then develop a concrete protocol by implementing (a special case of) the definition as a boolean circuit, and using efficient two-party computation techniques based on threshold homomorphic cryptosystems of [ST04] to evaluate it. Based on a detailed analysis of the efficiency of our protocol in terms of bandwidth, rounds of communication and computational overhead, we conclude that while this protocol is efficient enough for reasonably small policies, its overhead becomes prohibitive for larger ones. Therefore, we see our protocol more as a proof-of-concept, and as a benchmark against which the performance of future special-purpose protocols can be measured.

1.2. Related Work

Existing policy frameworks. The most common framework for privacy policy negotiation today is the Platform for Privacy Preferences Project (P3P) [W3C02]. A P3P policy consists of a number of attributes (e.g. the user's name, address,...), and the conditions tied to the user's willingness to reveal these attributes (e.g., it may not be forwarded to third parties, and has to be deleted once it is no longer needed). P3P was designed with two goals in mind. Firstly, it creates transparency about an organization's privacy policy. It is required that both the overall policy and some details are provided in a human-readable format, so every user can find out about the policy in an easy way.Secondly, it allows for automatic comparison of the policies. A user can thus define his personal privacy policy, and automatically get a warning if the policy of an organization he interacts with does not match (assuming the organization does provide a P3P policy [AT&02]).

To reduce the initial complexity of the standard, the current version of P3P deliberately left out any negotiation. Rather, the server simply reveals its policy, and the client then proceeds with the interaction, or not. In some sense, this does protect the client's privacy, but the server's policy is fully exposed.

Our negotiation protocol can be used within the context of P3P, though some practical restrictions may be necessary for efficiency reasons. The design of P3P is hierarchical in the sense that attributes (e.g. first name, last name) can be contained in other attributes (e.g. name). In the non-private case, this is not a problem – a policy may group attributes and thus not need to individually specify the preferences for each and every attribute. Our protocol, however, cannot efficiently handle such groups, as it makes the number of attributes too large to be practical. Also, P3P allows a policy designer to freely define attributes. This poses a problem for automated systems such as ours, as a user may not have predefined his preference on an attribute that the server defined.

In comparison with existing policy expression languages, some tradeoffs are required to satisfy the strict security requirements of a protocol leaking no information. We do need, for example, a pre-agreement on all data items and obligations that the policy can cover — it is not possible for a user to individually extend the preferences by new data items, as this would already give away information about his policy. Also, it is not clear whether hierarchical attributes can be used to improve efficiency of the protocol, since a difference in overhead of the protocol execution would leak information about the granularity of the parties' preferences.

Another popular framework for the definition of privacy policies is the *Enterprise Privacy Authorization Language* (EPAL) [BKBS04]. EPAL is designed as a back-end language to be used internally inside an enterprise, allowing to automatically enforce its privacy rules. As such, EPAL allows a fine-grained and flexible set of rules, which also represent the internal data flows within the corporation. While the possibility to express complex policies makes EPAL interesting for secret negotiations, its complexity makes it hard even to compare policies [BKBS04], let alone to negotiate them without revealing any information about the preferences. Some work is done on an automatic conversion from EPAL policies into a less precise format such as P3P, which would allow our protocol to be used in such a setting as well.

The theoretical model for policy negotiations proposed by Yu et al. [YWS03] focuses on allowing for a maximal independence in choice of strategy between the negotiating parties, without sacrificing efficient interoperability. They mainly consider a setting in which credential holders prove certified properties to a server, whereas we consider clients submitting unverified personal data, but their techniques can be applied to both types of negotiations. More importantly, they recognize the need to protect sensitive details of the parties' preferences [SWY01, YWS03], and propose protocols achieving this goal through a gradual release of requirements. Thereby, the disclosure of sensitive requirements is postponed until a certain level of trust has been established. It is hard to quantify however how much privacy this approach actually gives for general policies. In contrast, we employ cryptographic techniques guaranteeing that the only information leaked about the other party's preferences is the policy that was agreed upon.

Cryptographic protocols. The problem of private policy negotiation is a specific instance of secure two-party computation [Yao82, GMW87], which is the problem where two parties want to jointly evaluate a function on private inputs, while leaking no other information about their inputs than what is implied by the result. An efficient approach to secure two-party computation in the multi-party setting is to model the function as a boolean circuit, and to use a threshold homomorphic encryption scheme to evaluate it on encrypted inputs [AF90, FH96, CDN01, ST04]. (See Section 2 for more details on this approach.) We build on these results by implementing policy negotiation as a boolean circuit and evaluating it using the multiplication gates of [ST04]. The recently proposed double-homomorphic encryption scheme of [BGN05] cannot be applied to our setting because it can only handle circuits with a single level of multiplication gates. Though some attacks exist for the schemes underlying our (and in fact, most) zero knowledge circuits implementations with a dishonest majority [Cle86], they can be resolved by applying a slightly weaker model than usuall and carefull implemenatation, as the cheating party would clearly be exposed before any damage is done.

Private policy negotiation is also related to the problems of private matching and set intersection [FNP04], where two parties want to compute the intersection of their private datasets. Private set intersection could be used for a basic form of policy negotiation by letting the client's dataset contain the attributes that the client is willing to reveal, and letting the server's dataset contain the attributes that he wants to see. A matching policy exists if the intersection is equal to the server's dataset (which has to be determined using an extra zero knowledge comparison technique). Our protocol however supports more flexible preferences, allowing the client to express which attributes cannot be revealed *together*, and allowing the server to declare multiple combinations as sufficient for accessing the service. Moreover, to be useful in the model provided by existing privacy frameworks, we need to be able to model obligations. A user may well be willing to reveal data he otherwise would keep private if he is promised that it will be deleted within a week, or not forwarded to a third party.

Policy-based cryptography [BM05a, ARMLS04] is a way of enforcing need-to-know policies on the distribution of data, by allowing to encrypt data such that only users with certain roles can decrypt it. A trusted authority has the responsibility of issuing role certificates to the appropriate users. This line of work is complementary to ours, as it considers policy *enforcement* rather than negotiation. Moreover, their solution is not fully privacy preserving, as the ciphertexts leak information on the policies of the parties involved.

1.3. Further Use Cases

While our protocol was originally designed to negotiate privacy preferences, the approach can be used in various other settings where it is not sufficient to simply match attributes, but to allow users to negotiate a match based on more complicated preferences.

Non-humiliating dating. Assume two people want to find out whether they share common interests in order to decide if they should go on a date. To prevent humiliation, no interests are revealed unless shared by the other person, if any at all. In its simplest form, every party has a constant set of interests. A more complex setting is where each party has several sets of interests, corresponding to the different offers the party is willing to make. For example, a person may simultaneously seek quick affairs and more permanent relationships. Furthermore, he does not want a partner matching the "permanent relationship policy" to know that he was also looking for a quick affair, even if (or especially if) a match happens.

Business negotiations. The protocol may further be used for classical negotiation deals, i.e., for buying goods or services. Classical private negotiation systems are one-dimensional, i.e., both parties define an amount of money they are willing to spend or want to get, respectively, and the system tells them if a deal can happen (and potentially, for how much). However, most negotiations today have more facets. The seller may offer some discount if paid in cash, or if several items are bought, and the buyer may pay more if home delivery is ensured, or the warranty is extended. Assuming the number of options is not exceedingly high, our protocol delivers a practical way to privately negotiate the proper conditions.

1.4. Outline of the Paper

This paper is organized as follows. In Section 2, we present a brief overview of a homomorphic version of the El Gamal cryptosystem. Then we briefly discuss how threshold decryption and distributed key generation work in this setting, and how a multiplication on encrypted inputs is performed securely. Finally, we give efficient protocols for the secure evaluation of boolean AND, OR and NOT gates. In Section 3, we formally define the policy matching problem. In Section 4, we model policy negotiation as a function evaluation problem by describing a circuit consisting of boolean AND, OR and NOT gates implementing the negotiation process. We extend the circuit with *obligations* in Section 5, allowing a user to specify under which conditions he's willing to reveal each attribute. Again, a formal definition and a functional description in terms of a boolean circuit is given. Finally, we explain in Section 6 how the policy matching problem can be evaluated securely in a two-party setting, prove that the given solution is secure, and analyze the efficiency of our solution.

2. Secure Two-Party Computation

We have chosen to use the tools of [ST04] based on threshold homomorphic cryptosystems for various reasons. Firstly, threshold homomorphic cryptosystems allow for very efficient solutions of multi-party computation problems that are resistant against active adversaries [JJ00, DN03, CDN01]. There exist very efficient distributed key generation protocols for the discrete logarithm setting, making it attractive for ad-hoc contacts. Moreover, discrete-logarithm based protocols can be implemented using elliptic curves, resulting in smaller bandwidth requirements. Compared to the mix and match technique of [JJ00] it offers the same round complexity of $O(d)$, where d is the depth of the circuit being evaluated, but it is much more efficient for multiplications. (More precisely, the techniques developed in [ST04] are about ten times more efficient.)

2.1. Cryptographic Tools

Homomorphic encryption. Given an encryption function E, a public key p with corresponding secret key s, and a message m, we denote by $E_p(m)$ the encryption of m with the public key p. An encryption function E is called additively *homomorphic* if for all messages m_1 and m_2, we have $E_p(m_1 + m_2) = E_p(m_1)E_p(m_2)$. Well-known examples of homomorphic cryptosystems are the El Gamal cryptosystem [El 85] and the Paillier cryptosystem [Pai99]. In this paper, we mainly use the El Gamal cryptosystem. It has the advantage that key generation can be done very efficiently, and is therefore very well-suited for the P2P-setting. Most of the results we present, also hold however for the Paillier cryptosystem.

We describe briefly the homomorphic version of the El Gamal cryptosystem. Let $G = \langle g \rangle$ denote a finite cyclic (multiplicative) group of prime order q for which the Decision Diffie-Hellman (DDH) problem is assumed to be infeasible: given $g^x, g^y, g^z \in_R G$, it is infeasible to decide whether $xy \equiv z \pmod{q}$. This implies that the Computational Diffie-Hellman (CDH) problem[1] is infeasible as well. In turn, this implies that the Discrete Log (DL) problem, which is to compute $s = \log_g h$ given $h \in_R G$, is infeasible.

The public key of the El Gamal cryptosystem is an element $h \in G$ and the encryption of a message $m \in \mathbb{Z}_q$ is given by the pair $(a, b) = (g^r, h^r g^m)$ where $r \in_r \mathbb{Z}_q$. The secret key s is given by $s = \log_g h$.

Given the private key s, decryption of the ciphertext $(a, b) = (g^r, g^m h^r)$ is performed by first calculating $b/a^s = g^m$, and then solving for $m \in \mathbb{Z}_q$. This is done by restricting ourselves to messages m belonging to a sufficiently small domain $M \subseteq \mathbb{Z}_q$ which allows for exhaustive testing. Here, we take $M = \{0, 1\}$.

We define the multiplication of two ciphertexts (for the same public key) (a, b) and (a', b') by $(a, b)(a', b') = (aa', bb')$. It readily follows that this encryption scheme is additively homomorphic for this multiplication. Note that a message m has many encryptions under the same public key because of the randomness r involved. By multiplying $E_p(m)$ with $E_p(0)$ one obtains a new encryption of m under the same public key. Under the DDH assumption, this cryptosystem is semantically secure.

For ease of notation, we will use $[\![m]\!]$ to denote an encryption of the message m under some understood public key. In this notation we have $[\![x]\!][\![y]\!] = [\![x + y]\!]$ and $[\![x]\!]^y = [\![xy]\!]$.

[1] The Computational Diffie-Hellman problem is to compute g^{xy} given $g^x, g^y \in_R G$.

Threshold decryption. In an (n,t) threshold cryptosystem [DF90], the private key is distributed over n parties such that only coalitions of size at least t parties can decrypt the message hidden inside a ciphertext. In this paper we use a $(2,2)$ threshold cryptosystem where encryptions are computed with a common public key h but decryption is performed by running a protocol between the two involved parties. Every party holds a share $s_i \in \mathbb{Z}_q$ of the private key $s = s_1 + s_2 = \log_g h$, where the corresponding value $h_i = g^{s_i}$ is public. As long as both parties take part, decryption will succeed, whereas only one party is not able to decrypt successfully. Assuming that the parties have already obtained their shares of the secret key (through a distributed key generation protocol, explained below), decryption is performed as follows.For decryption of the ciphertext (a,b), the players P_1 and P_2 produce a decryption share $d_i = a^{s_i}$ $(i = 1,2)$ together with a proof that $\log_a d_i = \log_g h_i$. Assuming that both players produce correct decryption shares, the message is recovered from solving $g^m = b/a^s$ (where a^s is obtained as $d_1 d_2$) for m. Finally, it is checked whether $m \in \{0,1\}$. If this does not hold decryption fails. In case both parties need to obtain the decrypted value, the protocol has to be run in a *fair* way. A protocol for this is given in [ST04]. In case only one of the parties sends his share to the other party, a threshold decryption protocol with *private output* is obtained.

Distributed key generation. In order to set up the key generation in a P2P situation, the users have to run a distributed key generation (DKG) protocol. We describe very briefly a practical protocol [GJKR99] here. In the first step, both parties broadcast a Pedersen commitment $c_i = g^{s_i} h'^{r_i}$, with $s_i, r_i \in_R \mathbb{Z}_q$ along with a proof of knowledge for s_i, r_i. In the second step, both parties broadcast r_i along with a proof of knowledge of $\log_g h_i$, where $h_i = c_i / h'^{r_i}$. The joint public key is $h = h_1 h_2$, with corresponding private key $s = s_1 + s_2$. In many practical cases, a more lightweight one-round protocol[2] can be used. Then, both players broadcast $h_i = g^{s_i}$ and a proof of knowledge of s_i.

2.2. Secure Two-Party Computation from Homomorphic Encryption

In order to evaluate functions securely withstanding active attacks, zero-knowledge proofs are needed. We mention the most important proofs here for the discrete logarithm setting. One has Schnorr's protocol [Sch90] for proving knowledge of x given the public value g^x, and the extension of Okamoto [Oka92] proving knowledge of x, y given the common value $g^x h^y$. Furthermore there is the Chaum-Pedersen [CP92] protocol for proving knowledge of x given the common value $(a,b) = (g^x, h^x)$ (note that this gives a protocol for proving that a ciphertext (a,b) is an encryption of zero). Applying OR-composition [CDS92], one can distill from above mentioned proofs a protocol for proving that a ciphertext (a,b) is an encryption of a bit. We note that those proofs can be simulated and hence do not leak any information about the input to the protocol except what is revealed by the protocol itself. The most important ones are Schnorr's protocol [Sch90], Okamoto's [Oka92] and Chaum-Pedersen [CP92].

In order to be able to securely evaluate any circuit using an additive homomorphic encryption scheme, a protocol for secure multiplication is needed. We briefly remind the *private multiplier gate* and *the conditional gate* developed in [ST04]. Those protocols are simulatable even in the malicious case, and guarantee to both users that the computations have been performed correctly.

First consider the situation where the encryptions $[\![x]\!] = (a,b) = (g^r, g^x h^r)$ and $[\![y]\!] = (c,d)$ are given with player P_1 knowing x. Player P_1 computes on its own a randomized encryption $[\![xy]\!] = (e,f) = (g^s, h^s)[\![y]\!]^x$, with $s \in_R \mathbb{Z}_q$, using the homomorphic properties. Finally player P_1 broadcasts $[\![xy]\!]$ along with a proof showing that this is the correct output. which means that she proves knowledge of witnesses $r, s, x \in \mathbb{Z}_q$ satisfying $a = g^r$, $b = g^x h^r$, $e = g^s c$, $f = h^s d^x$.

Next, we consider a multiplication gate taking only encrypted inputs and for which the multiplier x is from a dichotomous (two-valued) domain, whereas the multiplicand y is unrestricted. It was called

[2] Although the trivial protocol allows one of the parties to influence the distribution of the public key h slightly, this need not be a problem for the application in which the key is used; see [GJKR03] for more details.

the *conditional gate* in [ST04].We present an explicit implementation for the two-party case. For sake of simplicity the formulation is done for the dichotomous domain $\{-1, 1\}$.[3]

Let $[\![x]\!], [\![y]\!]$ denote encryptions, with $x \in \{-1, 1\} \subseteq \mathbb{Z}_q$ and $y \in \mathbb{Z}_q$. The following protocol enables players P_1 and P_2, to compute an encryption $[\![xy]\!]$ securely.

1. Player P_1 broadcasts an encryption $[\![s_1]\!]$, with $s_1 \in_R \{-1, 1\}$. Then P_1 applies the private-multiplier multiplication protocol to multiplier s_1 and multiplicands $[\![x]\!]$ and $[\![y]\!]$, yielding random encryptions $[\![s_1 x]\!]$ and $[\![s_1 y]\!]$. Analogously, player P_2 broadcasts an encryption $[\![s_2]\!]$, with $s_2 \in_R \{-1, 1\}$. Then P_2 applies the private-multiplier multiplication protocol to multiplier s_2 and multiplicands $[\![s_1 x]\!]$ and $[\![s_1 y]\!]$, yielding random encryptions $[\![s_1 s_2 x]\!]$ and $[\![s_1 s_2 y]\!]$.

2. The players jointly decrypt $[\![s_1 s_2 x]\!]$ to obtain $s_1 s_2 x$. If decryption fails because $s_1 s_2 x \notin \{-1, 1\}$, the protocol is aborted.

3. Given $s_1 s_2 x$ and $[\![s_1 s_2 y]\!]$, an encryption $[\![(s_1)^2 (s_2)^2 (xy)]\!] = [\![xy]\!]$ is computed publicly.

2.3. Secure Evaluation of some Basic Gates

In Section 3 we turn the private policy negotiation problem into a problem of the secure evaluation of a function that can be described as a circuit consisting of basic *gates*, in casu NOT, OR and AND gates. For bits $x, y \in \{0, 1\}$, we use the shorthand notation $\neg x$ to denote the negation of x, we use $x \wedge y$ to denote the logical conjunction (AND) of x and y, and we use $x \vee y$ to denote the logical disjunction (OR) of x and y. We present protocols for the secure evaluation of those gates within the model of secure two-party computation; i.e. we consider two parties who evaluate these gates without revealing anything about their input (except the information that leaks from the output of the function).

We distinguish between protocols having only encrypted inputs and those having an encrypted and an unencrypted input. An unencrypted input refers to a private input of a user. The execution of the protocol should not reveal any information on this input. Encrypted inputs are inputs containing a message unknown to both players. Often they are the result from a previous (intermediate) computation whose result should remain unknown to both users. Clearly, the execution of the protocol should not reveal anything more on the message hidden by the encryption than what is revealed by the result of the protocol.

AND with Encrypted Inputs. Given two encrypted bits $[\![x]\!]$ and $[\![y]\!]$, the players run a conditional gate on those two inputs to compute $[\![x \wedge y]\!]$.

AND Gate with one Encrypted and one Unencrypted Input. Let x denote the private input and $[\![y]\!]$ the encrypted input which is available to both. The players run the private multiplier gate on inputs x and $[\![y]\!]$.

OR Gate with Encrypted Inputs. Given two encrypted bits $[\![x]\!]$ and $[\![y]\!]$, $[\![x \vee y]\!]$ is securely computed by running a conditional gate on the inputs $[\![x]\!]$ and $[\![y]\!]$. Then, using the homomorphic properties of the cryptosystem, they compute $[\![x \vee y]\!]$ as $[\![x + y - xy]\!]$.

OR with an Encrypted and an Unencrypted Input. Given unencrypted input x and encrypted input $[\![y]\!]$, the players run the private multiplier gate, yielding $[\![xy]\!]$. Then, the players compute $[\![x \vee y]\!] = [\![x + y - xy]\!]$.

NOT Gate on Encrypted Inputs. Computing $[\![\neg x]\!]$ given $[\![x]\!]$ is done by computing $[\![1 - x]\!]$ publicly.

[3] Domain $\{0, 1\}$ or any other domain $\{a, b\}$, $a \neq b$, can be used instead, as these domains can be transformed into each other by linear transformations: $x \mapsto a' + (b' - a')(x - a)/(b - a)$ maps $\{a, b\}$ to $\{a', b'\}$. These transformations can be applied directly to homomorphic encryptions, transforming $[\![x]\!]$ with $x \in \{a, b\}$ into $[\![x']\!]$ with $x' \in \{a', b'\}$.

3. Private Policy Matching: Definition and Approaches

3.1. Definitions

By the *preferences* of a user we mean a strategy defining which attributes (e.g. credit card number, address, ...) he is willing to reveal in order to gain access to certain service. The preferences of a server define the attributes he requires from a user before granting access to the service. Policy negotiation refers to the process of finding out whether a match exists between the attributes that the user wants to reveal and the set of attributes that the server requires. We say that a combination of attributes is a *matching policy* if it is acceptable to both the client and the server. We define the problem of policy matching more formally as follows.

Definition 3.1 Let $\mathcal{A}$ be a set of attributes, and let $\mathcal{S}$ be a totally ordered set of scores with least element 0. Preferences over the set of attributes $\mathcal{A}$ are described by functions $f, g : 2^{\mathcal{A}} \to \mathcal{S}$ that assign to each combination of attributes $A \subseteq \mathcal{A}$ a score $s \in \mathcal{S}$, indicating the client's willingness to reveal the combination of attributes A (in the case of client preferences f), or indicating the server's inclination to accept that combination of attributes as sufficient to access the service (in the case of server preferences g). A *matching function* $M : 2^{\mathcal{A}} \times \mathcal{S} \times \mathcal{S} \to \mathcal{S}$ assigns a *matching score* to a combination $A \subseteq \mathcal{A}$ based on A, the client's willingness $f(A)$ and the server's acceptance $g(A)$. A combination A is said to be a *matching policy* with respect to client preferences f, server preferences g and matching function M if $M(A, f(A), g(A)) > 0$. The *best matching policy* is the combination $A \subseteq \mathcal{A}$ for which $M(A, f(A), g(A))$ is maximal.

We introduced the set $\mathcal{S}$ to allow the expression of fine-grained preferences by assigning weights to sets of attributes. Throughout this paper however, we limit ourselves to the case $\mathcal{S} = \{0, 1\}$ and $M(A, f(A), g(A)) = 1$ iff $f(A) = g(A) = 1$, which corresponds to a client being either willing or unwilling to reveal a combination of attributes, a server either accepting a combination of attributes or not, and a match occurring whenever both parties accept the policy.

By *private policy negotiation* we mean a protocol between the client and the server during which they learn nothing about each other's preferences except whether a matching policy exists, and possibly what that matching policy is. In our model, we consider an active but static adversary who can corrupt one of both players and hence get access to all data of the corrupted player. External measures should be taken to prevent the client from extracting the server's preferences through repeated negotiations with different input preferences, e.g. by limiting the number of negotiations per client within a certain time interval or requiring human interaction.

3.2. Straightforward Approach

Let $\mathcal{A} = \{a_0, \ldots, a_{n-1}\}$ be the set of the client's attributes (e.g. $a_0 =$ "credit card number", $a_1 =$ "birth date", ...). If x is a bit string of length n, then we refer to the individual bits of x as $x_0 \ldots x_{n-1}$. To each $x \in \{0, 1\}^n$, we associate a set of attributes $A(x) = \{a_i \in \mathcal{A} : x_i = 1,\ 0 \leq i \leq n-1\}$. The client's preferences can be modeled as a boolean function $f : \{0, 1\}^n \to \{0, 1\}$, where $f(x) = 1$ if the client is willing to reveal the combination of attributes $A(x)$, and is 0 if he'd rather not reveal this combination. Likewise, the server's preferences can be modeled as a boolean function $g : \{0, 1\}^n \to \{0, 1\}$. A matching policy is an assignment $x \in \{0, 1\}^n$ such that $f(x) = g(x) = 1$.

The functions $f(x)$ and $g(x)$ are most naturally represented through their truth tables. Deciding whether a matching policy exists comes down to finding a row with a 1 in the output column of both truth tables. The most straightforward way to implement this approach as a boolean circuit is to let the client's input be a bit string $\hat{f} \in \{0, 1\}^{2^n}$ that is the output column of the truth table of $f(x)$, and to let the server's input be $\hat{g} \in \{0, 1\}^{2^n}$ being the output column of the truth table of $g(x)$. The circuit computes an index $i \in \mathbb{Z}_{2^n}$ such that $\hat{f}_i = \hat{g}_i = 1$. The size of this circuit (in number of inputs and number of gates), however, is $O(2^n)$, making it unsuitable for evaluation through secure two-party protocols.

3.3. Generating Subsets

A more compact yet quite natural description of the client's and server's policies can be obtained by observing that in most real-world cases, f is a monotonically decreasing boolean function, meaning that if $f(A) = 1$ and $B \subseteq A$, then also $f(B) = 1$. Indeed, if the client is willing to show the combination of attributes A, then it is natural to assume that he is willing to show any subset of these attributes as well. Likewise, it is easy to see that usually g is monotonically increasing, meaning that if $g(A) = 1$ and $B \supseteq A$, then $g(B) = 1$. Indeed, if showing attributes A is sufficient to access the service, then so should be any combination $B \supseteq A$.

Definition 3.2 Let $h : 2^{\mathcal{A}} \to \{0,1\}$ be a monotonically increasing boolean function. We say that $\mathcal{H} = \{H_1, \ldots, H_a\} \subseteq 2^{\mathcal{A}}$ is a *set of generating subsets* for h iff for all $A \subseteq \mathcal{A}$

$$h(A) = 1 \quad \Leftrightarrow \quad \exists\, i \in \{1, \ldots, a\} : H_i \subseteq A \ .$$

Note that since f is a monotonically decreasing function, $\neg f$ is a monotonically increasing function that is described through its set of generating subsets $\mathcal{F} = \{F_1, \ldots, F_a\}$. Essentially, the sets $F_1, \ldots, F_a$ are the *minimal* combinations of attributes that the client does *not* want to reveal together. Likewise, the function g is described through its set of generating subsets $\mathcal{G} = \{G_1, \ldots, G_b\}$, where the sets $G_1, \ldots, G_b$ are the *minimal* combinations of attributes that the server wants to see before delivering the service.

These generating subsets are not only a very compact representation of the client's and server's preferences, they are at the also a natural way of thinking about such preferences. The client for example may be reluctant to simultaneously show his credit card number and his mother's maiden name (the latter is sometimes used as a backup secret to reactivate lost cards), independent of other attributes he has to reveal in addition to that. Analogously, the server knows the minimal information that he needs from users (e.g. name and either email address or phone number), but he won't mind getting extra attributes

Using the notation of generating subsets, finding a match is equivalent to finding a set of attributes $A \subseteq \mathcal{A}$ such that

- $\forall\, F_i \in \mathcal{F} \ : \ F_i \not\subseteq A$, and
- $\exists\, G_j \in \mathcal{G} \ : \ G_j \subseteq A$.

Without loss of generality, we can assume that the matching policy is one of the server's generating subsets, if a match exists. (This is the match with the smallest number of shown attributes.) Therefore, we can write the condition for the matching policy A more compactly as the set of attributes $G_j \in \mathcal{G}$ such that $\forall\, F_i \in \mathcal{F} : F_i \not\subseteq G_j$.

3.4. Equivalence

Since truth tables and generating subsets are just different ways of describing the same preferences, there must be a relation between the two approaches. Indeed, we have that $f(x) = 0$ if and only if there exists a subset $F_i \in \mathcal{F}$ such that $F_i \subseteq A(x)$, and that $g(x) = 1$ if and only if there exists a subset $G_i \in \mathcal{G}$ such that $G_i \subseteq A(x)$. Casting this into a boolean formula gives

$$\neg f(x) = \bigvee_{F_i \in \mathcal{F}} \left(\bigwedge_{a_j \in F_i} x_j \right) \qquad g(x) = \bigvee_{G_i \in \mathcal{S}} \left(\bigwedge_{a_j \in G_i} x_j \right) \ , \tag{1}$$

showing that the generating subsets are actually the terms of $\neg f(x)$ and $g(x)$ when written in disjunctive normal form (DNF).

4. A Boolean Circuit for Policy Matching

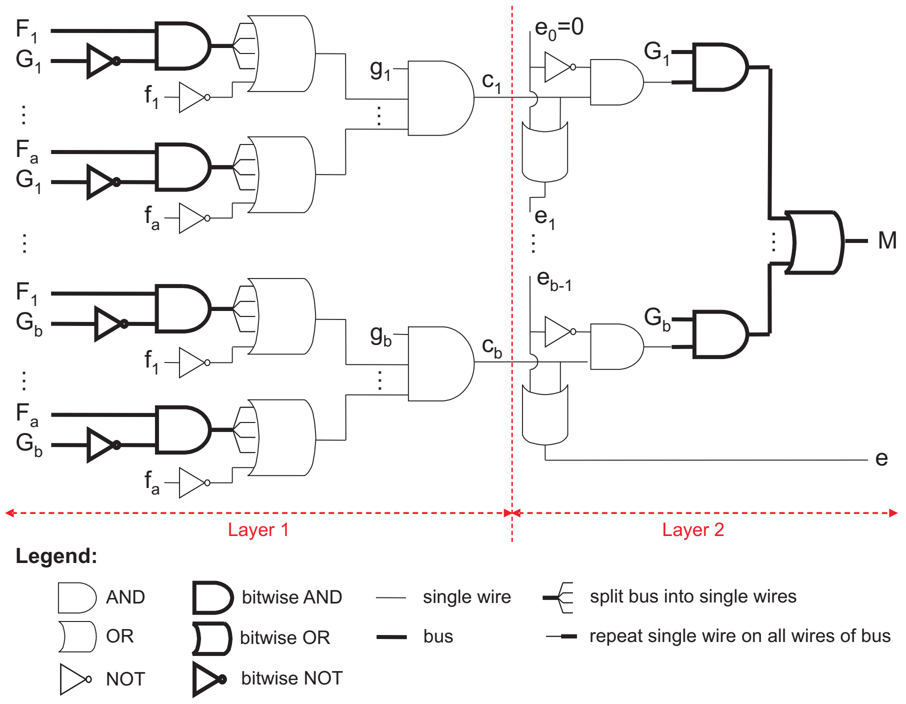

Figure 1: An efficient circuit for policy negotiation The client's inputs are generating subsets $F_1, \dots, F_a$, encoded as n-bit strings, and corresponding real-or-dummy bits $f_1, \dots, f_a$. The server's inputs are generating subsets $G_1, \dots, G_b$, strings, with corresponding real-or-dummy bits $g_1, \dots, g_b$. The output e indicates whether a matching policy exists ($e = 1$) or not ($e = 0$). The output M is an encoding of a matching combination of attributes, or 0^n if no such policy exists.

A boolean circuit implementing the generating subsets approach is given in Figure 1. The client's input consists of the generating subsets $F_1, \dots, F_a$ encoded as n-bit strings, where the j-th bit of F_i is 1 iff attribute $a_j \in F_i$. The server's input consists of the subsets $G_1, \dots, G_b$ in the same encoding. Since the values of a and b leak information about the complexity of the client's and server's preferences, the circuit needs to be designed for some fixed maximum values of a and b. Note that this leads to a worst-case scenario from the point of view of efficiency, but this is unavoidable as otherwise the run time would leak information about the preferences. The client and server assign arbitrary values to unused F_i and G_j entries, but distinguish "real" subsets from "dummy" subsets by setting the additional input bits f_i and g_j to 1 or 0, respectively. The output of the circuit is the encoding of a matching policy M, and a bit e indicating whether a matching policy exists ($e = 1$) or not ($e = 0$). We will see that when no matching policy is found, M takes the value $0 \dots 0$.

Gates with multiple fan-in in Figure 1 can be implemented as a cascade of binary gates (e.g. $x_0 \vee x_1 \vee x_2 \vee x_3 = ((x_0 \vee x_1) \vee x_2) \vee x_3$, or as a balanced tree of binary gates (e.g. $x_0 \vee x_1 \vee x_2 \vee x_3 = (x_0 \vee x_1) \vee (x_2 \vee x_3)$). Both options are equivalent in the total number of gates, but the latter option gives a better efficiency in terms of communication rounds, as we will see later. The thick lines in Figure 1 represent buses, which are essentially collections of parallel wires to carry words, rather than individual bits. Thick gates represent bitwise operations on words, e.g. the bitwise AND of n-bit words $x, y \in \{0,1\}^n$ is the n-bit word $z = (x_0 \wedge y_0, \ldots, x_{n-1} \wedge y_{n-1})$.

The circuit consists of two layers. The first layer checks, for all $j = 1, \ldots, b$, whether G_j is a suitable candidate for a matching policy, meaning that G_j does not conflict with any of the client's sets F_i. The wire labeled c_j in Figure 1 carries a one if G_j is a candidate policy, and a zero if not.

The circuit first computes, for all $i = 1, \ldots, a$ and $j = 1, \ldots, b$, whether the client's set F_i is "compatible" with the server's set G_j, meaning that $F_i \not\subseteq G_j$. This is equivalent to checking that there exists at least one attribute $a \in \mathcal{A}$ such that $a \in F_i$ but $a \notin G_j$. The output word of the bitwise AND gates in Layer 1 contains a one-bit for each such attribute. OR-ing the n bits of this word together gives 1 iff at least one such attribute exists. By including $\neg f_i$ in the OR, the output of the OR-gate is forced to 1 for dummy input sets F_i, representing the fact that policy G_j should never be rejected because of a conflict with a dummy set. The rightmost AND gates in Layer 1 decide, for all $j = 1, \ldots, b$, whether G_j is a candidate matching policy, meaning that it was "compatible" with all (non-dummy) client reluctancy sets. By including g_j in the AND gate, c_j is forced to 0 if $g_j = 0$, representing the fact that a dummy server set G_j can never be a suitable candidate.

The second layer finds the candidate policy with the lowest index and outputs it as the matching policy. To select the match with the smallest index, the circuit uses intermediate variables e_j that are 1 iff a matching policy exists among $G_1, \ldots, G_j$. The next value for e_j is computed as $\neg e_{j-1} \wedge c_j$, and the output bit $e = e_b$ is one iff a matching policy was found. AND-ing c_j with $\neg e_{j-1}$ ensures that the only non-zero bit coming out of any of the leftmost AND gates in Layer 2 is on the wire corresponding to the first match. The final gates of the circuit encode a matching policy onto the output bus M. Assume a match was found, and let j' be the index of the first match. Then the output of the j'-th bitwise AND gate in Layer 2 is $G_{j'} \wedge 1^n = G_{j'}$, while for all other AND-gates it is $G_j \wedge 0^n = 0^n$. The final bitwise OR-gate sets the output M to $G_{j'} \vee 0^n \vee \ldots \vee 0^n = G_{j'}$. If no matching policy was found, then M is set to 0^n, leaking no information about the server's preferences except the fact that no match exists with the given client preferences.

5. Policy Matching with Obligations

In this section, we extend the circuit to allow the client to express demands concerning certain attributes, such as that the data is deleted after a certain time, that it is not forwarded to third parties, or even to receive a discount in exchange for a certain attribute. The server expresses the promises he's willing to make for each attribute. A matching policy is then defined as a combination of attributes such that (1) they are deemed sufficient by the server to access the service, (2) the client is willing to reveal them, and (3) the server is willing to comply with the client's demands related to the revealed attributes. We extend Definition 3.1 with obligations as follows.

Definition 5.1 Let $\mathcal{A}$ be a set of attributes, let $\mathcal{S}$ be a totally ordered set of scores with least element 0, let f, g be functions describing the client's and server's preferences, and let M be a matching function as in Definition 3.1. Let $\mathcal{O}$ be a set of obligations. The client's *demand function* $d : \mathcal{A} \to 2^{\mathcal{O}}$ associates to each attribute a set of obligations that the client demands from the server when revealing that attribute. The server's *willingness function* $w : \mathcal{A} \to 2^{\mathcal{O}}$ maps an attribute to the set of obligations that the server is willing to respect for that attribute. We say that $A \subseteq \mathcal{A}$ is a *match* with respect to preferences f, g, matching function M, demand function d and willingness function w if $M(A, f(A), g(A)) > 0$ and $\forall\, a \in A \,:\, d(a) \subseteq w(a)$. The *best match* is the subset $A \subseteq \mathcal{A}$ for which $M(A, f(A), g(A))$ is maximal.

Again, we will only consider here the special case of Definition 5.1 where $\mathcal{S} = \{0,1\}$, where f and g are monotonically decreasing, respectively increasing, boolean functions, and where the result of the matching function $M(A, f(A), g(A)) = 1$ iff $f(A) = g(A) = 1$. Let $\mathcal{O} = \{o_0, \ldots, o_{m-1}\}$ be the set of promises that the client can demand for each attribute (e.g. $o_0 =$ "Delete after session", $o_2 =$ "Delete

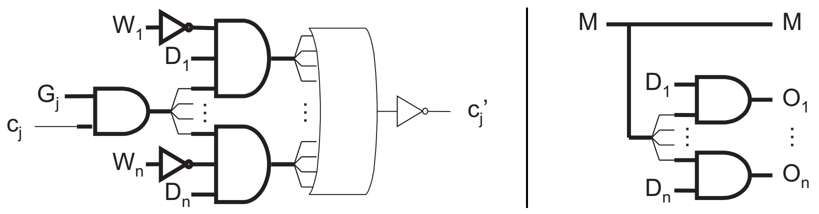

Figure 2: Extensions to the circuit of Figure 1 to support promises. The circuit on the left computes whether the server is willing (as defined by additional server inputs $W_1, \ldots, W_n$) to meet the client's demands (as defined by additional client inputs $D_1, \ldots, D_n$) for all attributes in a candidate policy G_j. The circuit on the right encodes the agreed-upon promises as part of the output.

after one year", o_3 = "Do not forward to third parties",...). The modifications to the circuit of Figure 1 are depicted in Figure 2. The client's demand function and the server's willingness function are described by additional input sets $D_i = d(a_i)$ and $W_i = w(a_i)$ for $i = 0 \ldots m-1$, respectively, encoded as m-bit strings. Apart from the matching policy M and a bit e indicating whether a match was found, the circuit now also outputs the obligations $O_0, \ldots, O_{n-1} \subseteq \mathcal{O}$ that the server has to adhere to for attributes $a_0, \ldots, a_{n-1}$.

The left circuit in Figure 2 is inserted b times between Layers 1 and 2 in Figure 1 on each wire c_j, $j = 1, \ldots, b$, replacing c_j with a bit c'_j before passing it to Layer 2. For each candidate policy G_j, this subcircuit computes a bit c'_j indicating whether the server is also willing to make all promises that the client requires for attributes in G_j. The first bitwise AND gate simply encodes G_j on the bus if G_j is a candidate ($c_j = 1$), or encodes a string of zeroes if not ($c_j = 0$). This encoding is split up in n separate wires representing whether attribute $a_i \in G_j$ or not, for $i = 0 \ldots n-1$. For each attribute a_i, the second column of bitwise AND gates outputs an m-bit word with a 1 on each wire representing an obligation $o \in \mathcal{O}$ that the client demands ($o \in D_i$) but that the server is not willing to make ($o \notin W_i$). If any such promise exists in any of the attributes in G_j, then policy G_j is not a match. This is exactly what is computed by the OR gate. The final NOT gate inverts this result so that $c'_j = 1$ if and only if G_j is a candidate match for which additionally the promises work out.

The right circuit in Figure 2 is to be appended to the right of the circuit in Figure 1. If attribute $a_i \in M$, then the bitwise AND gate encodes the client's demands D_i onto the output promises O_i, or it encodes the all-zeroes string if $a_i \notin M$.

6. The Private Policy Matching Protocol

6.1. Security

Security. In Section 3, we transformed the policy negotiation problem into a function evaluation problem to which each of the two parties provides its own private inputs (policies). Let us denote the corresponding function by C. First, the function C is described as a circuit consisting of AND, OR and NOT gates in Section 3. It was shown that those gates can be evaluated if addition and multiplication can be performed on encrypted inputs. Addition on encrypted inputs follows immediately from the homomorphic property of the used cryptosystem and the multiplication is done with the *conditional gate* of [ST04]. The function C is then privately evaluated by the following protocol $\mathsf{Func}_C(F_1, \ldots, F_a; G_1, \ldots, G_a)$.

1. Both players encrypt their inputs; i.e. they encrypt (bit by bit) the strings describing their generating subsets. They broadcast zero-knowledge proofs that they know the content of their encryptions and that the values they encrypted are bits.

2. They carry out all the gates of the circuit that describes the function f by using the secure

evaluation of the gates described in section 2.3. Gates that can be run in parallel will be securely evaluated in parallel, the others will be evaluated sequentially.

3. Finally, the output of the protocol is decrypted with a threshold decryption protocol.

Note that if the output should only be revealed to one of the players instead to both, then a threshold decryption with private outputs has to be used [ST04]. Finally, we mention that fairness can be achieved easily by using the fair decryption protocol developed in [ST04]. We have the following theorem.

Theorem 6.1 On input of the generating subsets $F_1, \ldots, F_a$ and $G_1, \ldots, G_a$ of the client and the server respectively, the protocol Func_C evaluates the private policy negotiation without leaking any additional information about $F_1, \ldots, F_a$ and $G_1, \ldots, G_a$.

Proof: Completeness of the Func_C protocol follows from the analysis in section 3 and of the construction of the gates in section 2.3. The fact that the Func_C protocol can be simulated follows from the following observation. The gates that are evaluated during the Func_C protocol consist on their turn of sequences of additions and multiplications. Hence it follows that the structure of the Func_C protocol follows exactly the structure of the function evaluation protocols in [CDN01, ST04]. This implies that the Func_C protocol can be simulated and hence leaks no additional information on its inputs $F_1, \ldots, F_a$ and $G_1, \ldots, G_a$. $\blacksquare$

Using the analysis in Section 5, which describes the extended policy matching problem with obligations as a function $\tilde{C}$, and the protocol $\mathsf{Func}_C(F_1, \ldots, F_a; G_1, \ldots, G_a)$ given above for secure evaluation of the function C, it is straightforward to write down a protocol for secure evaluation of the function $\tilde{C}$ i.e. policy negotiation with obligations. It readily follows from the description in Section 5 that the function $\tilde{C}$ can also be described by a circuit consisting of AND, OR and NOT gates. Hence, the proof that the associated protocol leaks no additional information on the player's inputs, is analogous to the proof of Theorem 6.1.

6.2. Efficiency

In order to assess the practical feasibility of our protocol, we estimated the overhead incurred by evaluating the circuits given in Figures 1 and 2 using the techniques laid out in Section 2. (A constant factor on the number of gates can probably be saved by applying a design automation tool such as Xilinx.) A number of representative values are given in Table 1. For both the basic circuit and the extended circuit with obligations, we computed the total amount of data sent over the network, the number of communication rounds, and the number of exponentiations to be performed by each of the participants. The actual values were obtained by observing that the evaluation of an AND/OR gate on encrypted inputs involves 11 exponentiations from each participant and 26 group elements to be communicated over the network in 2 rounds; that the evaluation of an AND/OR gate with one known input involves 4 exponentiations by one of the participants and 10 group elements to be communicated in a single round; and that the evaluation of a NOT gate comes practically for free (using the homomorphic properties of the encryption scheme). We were able to save on the number of rounds by evaluating independent gates in parallel, and by implementing bitwise gates on n-bit vectors using binary gates organized in a tree of depth $\lceil \log_2 n \rceil$, rather than in a cascade.

Asymptotically speaking, the basic circuit without obligations requires $O(abn)$ exponentiations to be computed and $O(abn)$ group elements to be communicated in $O(b + \log(bn))$ rounds. The circuit with obligations takes $O((a+m)bn)$ exponentiations and $O((a+m)bn)$ group elements in $O(b + \log(abmn))$ rounds.

			without obligations			with obligations		
n	a, b	m	bandwidth	rounds	exponentiations	bandwidth	rounds	exponentiations
10	5	10	235 KB	16	$4.11 \cdot 10^3$	1.15 MB	36	$2.15 \cdot 10^4$
50	25	25	24.0 MB	24	$4.29 \cdot 10^5$	88.3 MB	52	$1.62 \cdot 10^6$
200	50	100	373 MB	30	$6.66 \cdot 10^6$	1.97 GB	66	$3.76 \cdot 10^7$

Table 1: Efficiency estimates of our protocol for various parameter values when using the two-party computation protocol of [ST04] over 170-bit elliptic curves. We give the amount network traffic (bandwidth), the number of communication rounds and the number of exponentiations to be performed by each of the players for realistic values of the number of attributes n, the maximal number of client and server preferences a and b, and the maximal number of obligations per attribute m.

From Table 1, one can see that our protocol is practically feasible only for relatively simple preferences and/or resourceful environments. For larger parameter values, the overhead may become prohibitive. This is due to both the use of generic cryptographic primitives and the severeness of our privacy requirements. We think that our implementation, however, can still serve as a benchmark for protocols that use specialized techniques or relaxed privacy requirements.

7. Conclusion

We consider this paper as a first step towards privacy preserving negotiation protocols, whereas the main goal is to cleanly define the problem and demonstrate its feasibility. Consequently, this work raises a number of new questions that need to be addressed for the system to become practical.

One main issue is that our definitions of security are borrowed from the cryptographic community, and therefore are rather strict. For many applications, this level of security is too much, as it disallows a number of useful properties, such as open policies (i.e., allowing the user to introduce new attributes that are not in the standard), hierarchical policies (i.e., allowing the user to choose wether he wants a simple policy over general attributes, or a complex policy on specific attributes), and mixing data into the policy (e.g., express a statement like "If the user's age is smaller than 18, then the parents' address is required").

To resolve this question, a more realistic privacy metric is required. Defining such a metric is a challenging issue: on the one hand, it has to define real world privacy issues, on the other hand it should be sufficiently formal to allow for provably secure protocols. While there is some work on privacy metrics in the literature [SK03, DSCP02], the authors are not aware of any such metric that satisfies these conditions.

The second main issue is that our protocols use rather generic two-party computation. While the protocols are efficient enough to negotiate reasonably sized policies on Internet connected computers, large policies or devices with a small bandwidth (such as a cellular phone) will pose a problem. We expect that the cost of the protocols can be significantly decreased by using dedicated protocols instead of generic techniques.

Acknowledgements

The work of the first two authors was supported in part by the Concerted Research Action (GOA) Ambiorics 2005/11 of the Flemish Government and in part by the European Commission through the IST Programme under Contract IST-2002-507932 ECRYPT. Gregory Neven is a Postdoctoral Fellow of the Research Foundation – Flanders (FWO – Vlaanderen).

References

[AF90] Martin Abadi and Joan Feigenbaum. Secure circuit evaluation. *J. Cryptology*, 2(1):1–12, 1990. (Cited on page 4.)

[ARMLS04] Sattam S. Al-Riyami, John Malone-Lee, and Nigel P. Smart. Escrow-free encryption supporting cryptographic workflow. Cryptology ePrint Archive, Report 2004/258, 2004. Available from `http://eprint.iacr.org/`. (Cited on page 5.)

[AT&02] AT&T. Privacy bird, 2002. `http://www.privacybird.com`. (Cited on page 3.)

[BGN05] Dan Boneh, Eu-Jin Goh, and Kobbi Nissim. Evaluating 2-DNF formulas on ciphertexts. In J. Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 325–341. Springer, 2005. (Cited on page 4.)

[BKBS04] Michael Backes, Günter Karjoth, Walid Bagga, and Matthias Schunter. Efficient comparison of enterprise privacy policies. In *ACM SAC 2004*, pages 375–382, New York, NY, USA, 2004. ACM Press. (Cited on page 3, 4.)

[BM05a] Walid Bagga and Refik Molva. Policy-based cryptography and applications. In A. Patrick and M. Yung, editors, *Financial Cryptography 2005*, volume 3570 of *LNCS*, pages 72–87. Springer, 2005. (Cited on page 5.)

[BM05b] Adam Barth and John C. Mitchell. Enterprise privacy promises and enforcement. In *WITS '05: Proceedings of the 2005 workshop on Issues in the Theory of Security*, pages 58–66. ACM Press, 2005. (Cited on page 3.)

[CDN01] Ronald Cramer, Ivan Damgård, and Jesper B. Nielsen. Multiparty computation from threshold homomorphic encryption. In B. Pfitzmann, editor, *EUROCRYPT 2001*, volume 2045 of *LNCS*, pages 280–300. Springer, 2001. (Cited on page 4, 6, 14.)

[CDS92] Ronald Cramer, Ivan Damgard, and Berry Schoenmakers. Proofs of partial knowledge and simplified design of witness hiding protocols. In E. Brickell, editor, *CRYPTO 1992*, volume 740 of *LNCS*, pages 174–187. Springer, 1992. (Cited on page 7.)

[Cle86] Richard Cleve. Limits on the security of coin flips when half the processors are faulty. In *Proc. of the 18th ACM STOC*, pages 364–369. ACM Press, 1986. (Cited on page 4.)

[CP92] David Chaum and Torben P. Pedersen. Wallet databases with observers. In E. Brickell, editor, *CRYPTO 1992*, volume 740 of *LNCS*, pages 89–105. Springer, 1992. (Cited on page 7.)

[CV02] Jan Camenisch and Els Van Herreweghen. Design and implementation of the idemix anonymous credential system. In *Proc. of the 9th CCS*, pages 21–30, New York, NY, USA, 2002. ACM Press. (Cited on page 3.)

[DF90] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In G. Brassard, editor, *CRYPTO 1989*, volume 435 of *LNCS*, pages 307–315. Springer, 1990. (Cited on page 7.)

[DN03] Ivan Damgård and Jesper Buus Nielsen. Universally composable efficient multiparty computation from threshold homomorphic encryption. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 565–582. Springer, 2003. (Cited on page 6.)

[DSCP02] Claudia Díaz, Stefaan Seys, Joris Claessens, and Bart Preneel. Towards measuring anonymity. In Roger Dingledine and Paul Syverson, editors, *PET 2002* , volume 2482 of *LNCS*. Springer, 2002. (Cited on page 15.)

[El 85] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In G.R. Blakely and D. Chaum, editors, *CRYPTO 1984*, volume 196 of *LNCS*, pages 10–18. Springer, 1985. (Cited on page 6.)

[FH96] Matthew K. Franklin and Stuart Haber. Joint encryption and message-efficient secure computation. *J. Cryptology*, 9(4):217–232, 1996. (Cited on page 4.)

[FNP04] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. Efficient private matching and set intersection. In C. Cachin and J. Camenisch, editors, *EUROCRYPT 2004*, volume 3027 of *LNCS*, pages 1–19. Springer, 2004. (Cited on page 5.)

[GJKR99] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In J. Stern, editor, *EUROCRYPT 1999*, volume 1592 of *LNCS*, pages 295–310. Springer, 1999. (Cited on page 7.)

[GJKR03] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure applications of Pedersen's distributed key generation protocol. In M. Joye, editor, *CT-RSA 2003*, volume 2964 of *LNCS*, pages 373–390. Springer, 2003. (Cited on page 7.)

[GMW87] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or a completeness theorem for protocols with honest majority. In *Proc. of the 19th ACM STOC*, pages 218–229. ACM Press, 1987. (Cited on page 4.)

[JJ00] Markus Jakobsson and Ari Juels. Mix and match: Secure function evaluation via ciphertexts. In T. Okamoto, editor, *ASIACRYPT 2000*, volume 1976 of *LNCS*, pages 346–358. Springer, 2000. (Cited on page 6.)

[MPB03] Marco Casassa Mont, Siani Pearson, and Pete Bramhall. Towards accountable management of identity and privacy: Sticky policies and enforceable tracing services. In *DEXA 2003* , pages 377–382. IEEE Computer Society, 2003. (Cited on page 3.)

[Oka92] Tatsuaki Okamoto. Provably secure and practical identification schemes and corresponding signature schemes. In E. Brickell, editor, *CRYPTO 1992*, volume 740 of *LNCS*, pages 31–53. Springer, 1992. (Cited on page 7.)

[Pai99] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In J. Stern, editor, *EUROCRYPT 1999*, volume 1592 of *LNCS*, pages 223–238. Springer, 1999. (Cited on page 6.)

[Sch90] Claus-Peter Schnorr. Efficient identification and signatures for smart cards. In G. Brassard, editor, *CRYPTO 1989*, volume 435 of *LNCS*, pages 239–252. Springer, 1990. (Cited on page 7.)

[SK03] Sandra Steinbrecher and Stefan Köpsell. Modelling unlinkability. In Roger Dingledine, editor, *PET 2003* , volume 2760 of *LNCS*. Springer, 2003. (Cited on page 15.)

[ST04] Berry Schoenmakers and Pim Tuyls. Practical two-party computation based on the conditional gate. In Pil Joong Lee, editor, *ASIACRYPT 2004*, volume 3329 of *LNCS*, pages 119–136. Springer, 2004. (Cited on page 3, 4, 6, 7, 8, 13, 14, 15.)

[SWY01] Kent E. Seamons, Marianne Winslett, and Ting Yu. Limiting the disclosure of access control policies during automated trust negotiation. In *Proceedings of the Network and Distributed System Security Symposium (NDSS 2001)* . The Internet Society, 2001. (Cited on page 4.)

[W3C02] W3C. The platform for privacy preferences 1.0 (P3P1.0) specification, 2002. `http://www.w3.org/TR/P3P/`. (Cited on page 3.)

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations. In IEEE, editor, *Proc. of the 23rd FOCS*, pages 160–164. IEEE Computer Society Press, 1982. (Cited on page 4.)

[YWS03] Ting Yu, Marianne Winslett, and Kent E. Seamons. Supporting structured credentials and sensitive policies through interoperable strategies for automated trust negotiation. *ACM Trans. Inf. Syst. Secur.*, 6:1–42, 2003. (Cited on page 4.)