Verifiable and Revocable Expression of Consent to Processing of Aggregated Personal Data

Henrich C. Pöhls

University of Passau, Institute of IT-Security and Security Law (ISL), IT-Security
Innstr. 43, 94032 Passau, Germany
henrich.poehls@uni-passau.de

Abstract. We have identified the following three problems for the processing of aggregated personal information with respect to privacy preferences: Unverifiable proof of consent, unverifiable proof of consent for aggregated personal data, and no verification if the consent is still established. We constructed a solution based on a hash tree structure and digitally signed only the hash tree's root value. Thus, a verifiable signature can be retained even if data items are omitted and a valid signature serves as signal of consent. To re-assure that no change of consent has taken place we propose the use of certificate revocation mechanisms. As a side-effect these mechanisms allow to maintain a record of personal data usage and thus creates a win-win situation for both parties involved.

1 Introduction

Looking at the automatic processing of aggregated data the question of privacy shall always be raised. Following the definition given by the EU [5] data items that are personally identifying will be called *personal data* and "shall mean any information relating to an identified or identifiable natural person (*data subject*);" [5]. To be legally allowed to process this personal data the data subject needs to express his consent to the processing: "the *data subject's consent* shall mean any freely given specific and informed indication of his wishes by which the data subject signifies his agreement to personal data relating to him being processed." [5]

In digital systems this consent can be expressed by the data subject's election to "opt-in" or "opt-out" [10] of certain processing. This consent is usually expressed at the time the user provides the service with his personal data. Services collecting personal data state in their *privacy policies* how the service will handle the submitted personal data. As these services process data we call them *processors*. The processor's privacy policy can be machine understandable, like P3P [21] for websites. Most of the time we find services constructed in such a way that users not consenting to the service's privacy policy will not be able to participate[1]. Thus, a direct relationship between the data subject and the data

[1] Often found: "By using the service you consent to the collection and use of information about you in the ways described in this Privacy Policy."

L. Chen, M.D. Ryan, and G. Wang (Eds.): ICICS 2008, LNCS 5308, pp. 279293, 2008.

processor exists. A direct relationship is the simplest case and transmits the data subject's expression of consent directly to the processor.

Besides the processor's privacy policy the processing of personal data can be restricted by the data subject. The restrictions and constraints under which the data subject allows the processing will be called *privacy preferences*. Most of the time the service's privacy policy is equal to the data subject's privacy preferences. The processor's privacy policy can be seen as a suggestion for privacy preferences. The data subject accepts them as his own by expressing consent. Hence, the processor dictates the restrictions instead vice versa. We do not further elaborate on this problem, but the proposed solution allows the data subject to set forth his privacy preferences.

In case of a dispute, the processor is obliged to present a proof that consent was established, as "personal data may be processed only if the data subject has unambiguously given his consent" [5]. With no verifiable paper-trail, i.e. no paper lottery ticket with a ticked "opt-in" box, the service retains no direct proof of established consent. Implicitly, the service's sign-up process and the privacy policies the data subject agreed to, in order to use the service, at the time of data submission can be used. In order to serve as a proof that consent was established, this process has to be documented, time stamped, and attested by a trusted third-party. Even in the simple case of a direct relationship, constructing a verifiable proof of the data subject's consent to data processing does not exist in most environments and requires trusted third-parties.

In a loosely connected, decentralized environment, with web services, agents, or Web 2.0 [17], a direct relationship is often not existing or its establishment would be a hindrance to the user. With compound services, like most web services, the original processor engages additional sub-services to carry out the processing. Each sub-service, if respecting privacy, would request a proof of established consent before processing the given personal data. With the advent of semantic applications on the semantic web [1] the exchange of such data will increase. Services will combine or re-combine data items from different sources and construct new data sets, a process we denote by *aggregation*. An aggregator should, under circumstances set fourth by the data subject, be able to derive the consent for the processing of aggregated personal data from the initial consent established for the source data.

Personal data is also exchanged between services who have no direct relationship to the data subject. On the other hand, the data subject can rectify, erase or block the processing [5]. All these actions will *vanish the consent*. Until the time personal data is processed, the data subject might have vanished his consent to processing, rendering any further processing "unconsented" and thus inducing possible legal challenges if processed. At most processing services, that have a direct relationship to the data subject, are aware of this state change in the consent from *established* to *vanished*. However, a service with no direct relationship is left unaware that the data subject's consent has vanished. Vanishing consent removes the service's ability to rightfully process and use the personal data held by it any longer. A suggested approach is to time limit a given con-

sent, but most often the time of state change can not be predetermined, i.e. a change of address. Thus, services need a way to automatically determine that the consent is still established.

To summarize, we have identified the following three problems:
No possibility for third-parties to verify:

(i) data subject's consent to personal data processing,
(ii) data subject's consent to aggregated personal data, and
(iii) that the data subject's consent is still established.

Our approach presented in this paper solves these three problems by retaining a proof of established consent with the data. This proof is verifiable by third-parties as well as by the data subject, and shows what privacy policy and combination of data items the data subject consented to. Additionally, a once established consent can be vanished, to indicate that the processing once consented to is no longer consented to.

The rest of the paper is organised as follows: Starting with a short scenario in Sect. 2, we will define more terms and clearly outline the problems in Sect. 3. We will present our solution in Sect. 4 and show how existing mechanisms from public key infrastructure (PKI) can be used to technically implement the concept. We thoroughly discuss our solution in Sect. 5, including a discussion on the performance and a discussion of existing work. We also show the security of our solution in the presence of an attacker. Finally, we conclude in Sect. 6.

2 Scenario

In the given scenario, Mr. Luxe, is assumed as the data subject. Mr. Luxe has a profile on a social network service (SNS), in the scenario called *Soc*. *Soc* is the service Mr. Luxe has a direct relationship with, and he has consented that *Soc* uses his personal data like email and relations as set forth by *Soc*'s privacy policy. Now *Soc* is no walled garden and data items, like the list of Mr. Luxe's friends Alice and Bob, are accessible by sub-services[2]. Apart from keeping his social relations on *Soc*, Mr. Luxe is a registered user on the website *example.org*. *example.org* allows registered users to upload examples, but Mr. Luxe only seldomly visits it. However, his friend Alice is an active user of *example.org* and regularly uploads new examples.

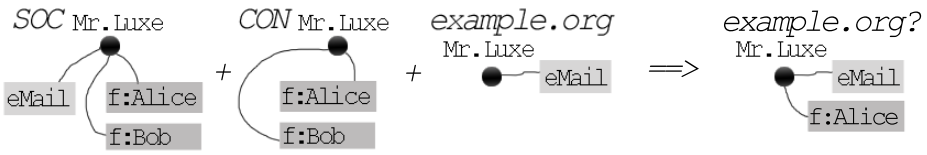


Fig. 1. Scenario data sets representing Mr. Luxe

[2] In SNS terminology often called "apps","widgets" or "gadgets"

A sub-service, called Con, now offers the functionality to indicate which known friends are also registered members on a website[3]. To enhance the user's experience, *example.org* incorporates Con's service to offer additional notifications whenever friends are active on the website, for example sent an email when new content is added by a friend. From the website's point of view both, Con and Soc, are third-parties. The website knows its list of registered users, Mr. Luxe and Alice are both in this list, but *example.org* does not know their social relations. Now through the use of Con, *example.org* is given the information that Alice is a friend of Mr. Luxe on Soc. From the registration and during the use of *example.org*, the website has collected its own set of information about Mr. Luxe and Alice, for example the email address and their browsing habits.

Some questions that arise are: Has Mr. Luxe consented to an aggregation of *example.org*'s data with data received from Con? Can *example.org* sent him emails whenever Alice posts something on the site, to boost his visits? Can Con, as a service in the middle, be presented with a consent to process and transfer the data that Mr. Luxe and Alice are friends without having a direct relationship with them? In the rest of the paper we will look more closely at the identified problems and present our solution that allows to answer the questions.

3 Semantical data, privacy preferences vs. policies, and the identified problems

The scenario indicates the need to flexibly allow the use of different portions of the personal data D. So, we split the personal data into atomic elements that comprise D, called *data items*, denoted as d_i. Each data item covers one semantic concept and can not be split further, i.e. a field in a data base. Hence, D is defined as a set of all d_i. Every single data-item is identifiable by a name $\{d_i\}_{ID}$ that carries the semantic meaning. The data item's value is denoted $\{d_i\}_{VAL}$. A data graph [19] captures the data items and the respective privacy preferences. The privacy preferences that are applied to a data item d_i are denoted as $ppref_i$. Each $ppref_i$ can contain one or more privacy preference expressions $pref$ to express the privacy preferences for d_i. All the data subjects privacy preferences for D are denoted as $PPref_D$ being the set of all the $ppref_i$.

How the privacy preferences expressions $pref$ and privacy policies $PPol$ are codified and later enforced is not within our scope, the only constraint is that it needs to be interoperable among the involved parties. Thus, we assume that the involved parties share a common vocabulary for the variable names as well as for the privacy preferences. So variables with the same semantic meaning can be identified, and the terms expressing the privacy constraints are understood by all parties. Ontologies can be used to establish this common vocabulary among all the participating parties. Hence, we assume that for variables and preferences a single ontology, including domain-dependent and domain-independent ontologies, exists [19].

[3] This sounds familiar to a service called Friend Connect [6], but they differ technically and policy-wise.

A boolean function *match(PPref,PPol)* allows to check if the data subject's preferences are fulfilled by a service's policy, for example by applying policy subsumption as described by Squicciarini et al. [19]. With this function a service can check which available third-party service's policy would comply, thus being able to hand personal data to complying sub-services only. Here the first problem becomes visible: The sub-service that receives the personal data lacks the ability to verify the data subject actually expressed his consent to process this data. We will now look at each problem in more detail.

3.1 No verification of consent to data processing

As mentioned, the sub-service is a third-party. Since it has no direct relationship with the data subject, it has no way of being provided with the user's consent through a sign-up process. The function *match* only allows to check if the provided privacy preferences can be fulfilled. Our goal is to ensure that the sub-service will still be able to verifiable check the data subject's consent before processing.

The proposed approach provides services with an additional boolean function *verify*. A positive result of $verify(S, D, PPref_D, PPol)$ allows to have confidence that processing the data items of D following the privacy policy $PPref_D$ is consented to by the data subject S. Vice versa, a negative result would indicate that the processing is not be consented to. With *verify* we free the sub-service from the need to trust the upper layer service, the service he got the data from. For a processor, regardless on what layer, it shall not matter if it has gained the data through a direct relationship, gained the personal data by transfer, or by other means like harvesting openly available sources. In all cases a positive result of *verify* assures the processor that processing under its *PPol* is consented to by the data subject.

3.2 No verification of consent to aggregated data processing

The second problem comes from the aggregation of personal data items by a service that received them through different sources. First we also want to allow the opposite of aggregation to happen: Fragmentation. Fragmentation of data takes place rather often, as only some portions, some data items, of the complete data set are shared. From a privacy point of view sharing of a minimal sub-set of data items is preferred. The verification of a fragment given a data set A from the data subject S works as follows: If $verify(S, A, PPref, PPol)$ is positive then $verify(S, C, PPref, PPol)$ shall also yield a positive result iff $C \subseteq A$.

The opposite is the aggregation of two data sub-sets. For example the original and complete data set contains three data items $A = \{a_1, a_2, a_3\}$. Two sub-sets $\{a_1, a_2\}$, $\{ppref_1, ppref_2\}$ and $\{a_1, a_3\}$, $\{ppref_1, ppref_3\}$ are aggregated. As the aggregated data set will contain all data items, the verification of the aggregated set $verify(S, \{a_1, a_2, a_3\}, \{ppref_1, ppref_2, ppref_3\}, PPol)$ yields the same verification outcome as $verify(S, A, PPref_A, PPol)$. More details follow in Sect. 4.

3.3 No verification of consent state changes

The last problem we identified was that the state of the consent expressed for the processing can change over time. Data fragmentation, aggregation, and the involvement of sub-services lead to the distribution of personal data among different services. Thus, re-use of personal data will occur over time. On the other hand, data items move virtually away from the data subject. Hence, each service needs the ability to gain re-assurance that an initially given consent is still established.

Three states are possible: *No consent*, *established consent*, and *vanished consent*. Two state changes are expected: From no expression to established, and from established to vanished. To initially establish consent the verifiable proof of consent, together with the data subject's privacy preferences is added and distributed along with the data set. To vanish consent it is not enough to just remove or strip the proof of consent completely. A retained expression of once established consent must serve as a proof for all processing occurred previous to the state change. Therefore, the two states established consent and vanished consent are verifiable, while the state no consent can not be verified. As data subjects are often unaware of their personal data' flows, each service shall be responsible to check that the consent has not vanished at processing time. This check must be carried out during the evaluation of the *verify* function. Of course, a status check involves additional communication between the service and the data subject or a trusted-third party acting on his behalf. This is comparable to a certificate revocation checking mechanism [8], and we will indeed propose to use a solution for handling the consent state changes technically based on certificate revocation.

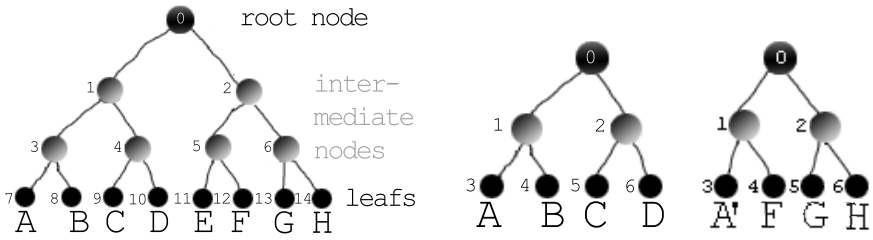
4 Solution

We will now explain our proposed solution allowing to retain a verifiable expression of established consent, that can be later vanished, in more detail. To represent the data D, we choose a tree. Each leaf contains a tuple of a single data item d_i and its privacy preferences, so $\langle d_i, ppref_i \rangle$. For this tree we compute a hash tree, comparable to Merkle [13] hash tree. Starting from the leafs containing the tuples, each intermediate node is a hash of the concatenation of its children nodes. Finally the root node will be associated a hash value depending on: All $\langle d_i, ppref_i \rangle$, the tree's structure, and the association of tuples to leafs. We will then apply a digital signature scheme to just sign the root hash. We assume a public key to be bound to a data subject S, thus we are able to associate S with S's signature, for example by a trusted public key certificate. This protects the integrity and offers non repudiation with respect to identifying the key-pair used for signing. The leaf's and the intermediate node's hash values are then discarded.

In difference to a classical hash over $D||PPref$ a tree-based hash allows omission of sub-trees. To omit a sub-tree rooted at a given node, while still being able to re-compute the same root hash as in a tree without omissions, the node's

hash value needs to be supplied as substitution for the omitted sub-tree. For example the leafs A and B in Fig. 2(b) can be removed if the hash of node 1 is given, we call this a *substitution hash* of 1. A signature over a tree rooted at node X is denoted by $signed(X)$. So additionally to the tree's structure the nodes within the tree need to be identifiable and the association of tuples needs to be communicated. We will assume that this information is known. For brevity and better readability we use a shorthand notation: To indicate that only the data items A and B of the tree from Fig. 2(a) are present we write $A, B; 2, 4 + signed(0)$, instead of $(7 : A), (8 : B); (2 : \textit{substitution hash of 2}), (4 : \textit{substitution hash of 4}) + signed(0) + \textit{treeinfo}$.

We will call this data tree, accompanied by the digital signature and additional information, a *signed data tree*.



(a) One data tree with 8 leafs (b) Two smaller trees

Fig. 2. Example binary trees, each leaf represents a *tuple* $\langle d_i, ppref_i \rangle$

.

4.1 Retaining a verifiable expression of consent

Our signal to indicate consent is the digital signature and verifying it works as follows: First the *verify* algorithm recomputes the hash tree root, using supplied substitution hashes where necessary. Note, we strictly forbid substitution hashes on paths leading to a not omitted tuple as this introduces ambiguities. So $A, B, C; 2, 4 + signed(0)$ would not yield a positive verification. To indicate the omission of D one needs to supply $A, B, C, h(D); 2 + signed(0)$. If the root hash can not be constructed verification fails. Second the digital signature is verified as usual according to the digital signature scheme. A positive signature verification for a tree indicates the consent to use any combination of the data items obeying the data item's privacy preferences that are part of the tree. So, with appropriate substitution hashes omissions are consented to.
Example: When node 2 in Fig.2(a) is signed then any combination of the tuples E, F, G, and H are consented to by the signer.

4.2 Allowing aggregation, while retaining a verifiable expression of consent

Verifying the aggregation of two verifiable signed data trees is only sensible, if the signers are the same. You can of course aggregate arbitrary data items, but a verifiable signature means that the data subject has consented to this aggregation. In other words, trying to verify the aggregation of verifiable data items from Alice with verifiable data items from Bob means asking if Bob consented to the aggregation with items of Alice and vice versa. As Bob does not need to know Alice, this aggregation of verifiable data items only makes sense if they are signed by the same key or the same identity.

We denote aggregation by $\otimes$. *Example from Fig.2(a):* E,G;$h(F)$,$h(H) + signed(2) \otimes F = E, F, G; h(H) + signed(2)$ obviously verifies. Aggregation is especially obviously consented to when we can find a "smaller" singed data tree with a hash-value equal to a substitution hash in a "bigger" signed hash tree. *Example from Fig. 2(a):* We assumed A, B, C, and D being of equal value in the trees. Aggregating a big, partially known, but root signed tree with a smaller, incomplete signed sub-tree: $E, F, G; h(H), 1 + signed(0) \otimes A, B, D; h(C) + signed(1) = A, B, D, E, F, G; h(C), h(H) + signed(0)$. This result will verify, thus an aggregator, still without knowing the real values of C and H, is able to reproduce a verifiable signature of a larger tree after the aggregation.

An aggregator does not need a signed data tree to add an omitted data item d_x. If he knows the $ppref_x$ he can check prior to aggregation if $hash(\langle d_x, ppref_x \rangle)$ corresponds to a substitution hash. If it corresponds, he can add the tuple during an aggregation and *verify* will yield a positive result. Note, aggregating will only yield a positive verification if the data items added during aggregation directly match a substitution hash or if the aggregator is able to provide missing substitution hashes. *Example from Fig.2(a):* To verifiable aggregate $A, B; 4, 2 + signed(0)$ and C the aggregator needs $h(D)$.

A data owner, wanting to allow future aggregation of already known data items, does not need to release all the data item's values $\{d_p\}_{VAL}$. Instead, he can release $hash(\{d_p\}_{VAL})$, $\{d_p\}_{ID}$, and $ppref_p$. Doing so he signals his consent to possible future addition of the omitted values under the already stated privacy preferences.
Example from Fig.2(a): E,F;$h(G)$,$h(H) + signed(2)$ allows to later add G or H. In contrast E,F;$6 + signed(2)$ only verifies if either G and H, or their substitution hashes can be provided. Hence, on signature generation the data subject can willingly express his consent to foreseen future aggregation, by supplying substitution hashes of leafs as place holders.

As the signature protects the integrity, aggregation with changed data items or changed privacy preferences will never yield a verifiable signature.
Example from Fig. 2(b): We assume the same signer for both trees. The aggregator's goal is to combine data items from the left tree A, B with data items from the right tree A', E. If $A = A'$, and the aggregator is presented with the inputs $B, C, D; h(A) + signed(0)$ and $A', E + signed(1)$, the aggregation $A', B, C, D + signed(0)$ will verify. All other combinations, for example B, E will

not have a path to a common signed node, and will not verify. So to not consent to a combination, the data subject simply puts these data items into separate trees.
Example: To give no consent to the aggregation $(B, C, D) \otimes (E, F, G)$ while still allowing both of them to be combined with A results in the two signed trees depicted in Fig. 2(b) with $A = A'$.
Thus, to forbid certain aggregation the data subject needs to make sure that these data items do not end up in the same signed tree. Generating a different signed data tree for each different service the data subject interacts is quite natural. Note, our approach does not demand that a data item that is added during a consented aggregation must initially come from a signed data tree.
Example: Having $B, C, D; h(A) + signed(0)$, A can be added without respect to its origins. This assumes that the aggregator knows A's value, but also the identifier $\{A\}_{ID}$, and the privacy preferences $ppref_A$.

4.3 Allowing status changes of consent by using PKI mechanisms

To enable a status check the following information is additionally signed: A time period *established-until* and information about a service to check the consent's actual status, named *status provider*. At the time of establishing consent, before signature generation, the data subject needs to encode the information for the status provider and optionally set the time limit. Thus, the data subject defines how the status provider can be contacted. Using time periods limits the time in which consented use can appear. If the time specified in *established-until* is reached, the consent is no longer established but vanished.

The status provider provides a function *check* to processors and third-parties. If the consent is still established, *check* yields a positive outcome. If *check* returns a negative result, this means that the consent has vanished. As a extension we envision to additionally provide information why the consent has vanished. The data subject controls the outcome of the *check* function. Using the status provider the processor queries the status of a given consent as it carries out the $verify$ procedure. As only the data subject shall be allowed to give an authoritative answer, the outcome of *check* must be signed and contains a time-stamp. To retain a verifiable proof that, at the time of processing, the consent was still established, the processor saves the signed and time-stamped answer for his record.

This sounds familiar to the information and mechanisms usually found in public key certificates. In X.509 public key infrastructures (PKI) the revocation status of certificates is queried through certificate revocation lists (CRL) [7] or the online certificate status protocol (OCSP) [16]. A PKI can be used to determine the trust of the binding between the data subject's identifier and his public-key. Further, X.509 certificates and the revocation mechanisms can be facilitated to transport the information in our solution: The root hash value and tree information can be stored inside a X.509 certificate. Our previous work [18] showed the applicability for a similar use. Such a certificate will carry the status provider information, as a CRL distribution point. The time period will be stored

as the certificate's validity period. The data subject's public-key, the root hash, and additional information can be stored as well. The certificate would be issued by the data subject, thus digitally signed using the data subject's private-key. So all the relevant information is protected and it can be revoked without affecting the bond between the data subject and his key-pair.
Example: To participate in a raffle we consent that name and postal address are processed in order to send us potential winnings, but as we know the draw takes place in May, we could limit the established consent to the end of June. The vanished consent rendering all uses later than June "unconsented". Note aside, this could also be defined using privacy preferences that handle timing.

The data subject controls the information which status provider a processor uses to query the status of consent. Thus, logging revocation status checks allows the data subject to see for which complete signed data trees verification is requested. In other words, *check* is a call-back to the data subject, allowing him to see which of his data sets is in question of being processed. Allowing to only log on the level of data sets can be seen as a compromise, as it leaves some privacy for the data processor. Nevertheless both parties have a gain from using the status provider: The processor gains a re-assurance in the proof of the consent, and the data subject gains information about the re-use of his personal data.

4.4 Security in the presence of an attacker and enforcement through detection

We assume that strong asymmetric cryptography and cryptographically sound hash functions are used to generate the signature and the hash values. So an attacker without knowing the data subject's private-key can not modify or reproduce the expression of consent codified in the digital signature without knowing the appropriate values. However, there is no protection against "unconsented" data processing. Our approach deliberately chooses not to impose additional access control restrictions to the personal data. An attacker can simply ignore or change privacy preferences during processing, but he will not be able to present a proof of consent that yields a positive verification outcome. Thus, "unconsented" data processing destroys the verifiable proof of consent. With a system as ours in use, personal data without a verifiable consent has only little business value, because it can not be used further used to interact with the data subject or released to third-parties without the loss of consent going undetected. Thus, an attacker must fear legal implications or looses reputation when using or passing on unverifiable personal data.

5 Discussion

5.1 Related Work

Privacy compliant processing can either be guarded through access control and checked upfront or compliant processing can be checked and verified afterwards.

Our work falls into the second category, but we see the two approaches as complimentary not as mutually exclusive. Examples of upfront checking approaches that can easily be augmented by our approach are work by Hutter et al [9] and Squicciarini et al. [19]. Hutter et. al. showed that while planning the composition and execution of compound web services the data owner's preferences with respect to privacy can be matched against each service's privacy policy, resulting in composition plans that use only web services that do not violate the given data's security policy. They use data flow analysis to see if the service plan respects the given privacy constraints and enforce this in a "trusted" composition engine. Squicciarini et. al. [19] defined in their work how data handling preferences set forth by the data subject can be matched against the policies of processing services. Their approach bases on policy subsumption. This allows to check whether the data can be handed over to the next service, because its handling preferences comply with the policies or not. Both offer no retainable proof of consent to benign parties involved.

In the case of checking privacy compliance afterwards the term of information accountability has been brought up by Weitzner et. al. [23]. They discuss the legal and technical framework to enable accountability for privacy. In another work Weitzner [22] showed how a simple logging facility, invoked whenever data is processed, can already help to detect privacy violations. Their work lacks a proof that processors, data subjects, and third-parties alike could verify.

Related in the field of aggregation is work done Devanbu et al. [4], Bertino et al. [2], and Carminati et al. [3]. Devanbu et al. use Merkle hash trees to verify the completeness of answers on queries. Their focus on verifying partial trees would be applicable for verification, but does not touch aggregation processes. The latter works [2] [3] protect data in transit without the need of trustworthy publishers, thus are more focussed on confidentiality.

We propose the use of digital signatures and suggest reusing existing PKI mechanisms. These mechanisms are well studied, as also are their overheads [15]. Our approach does not restrict the choice of the digital signature scheme. Schemes that need less resources for revocation could be used, for example Le et al. [11] proposed using a reverse chain of forward secure signature (FSS) in order invalidate certified credentials with minimal overhead from CA or OCSP/CRL involvement. Invalidating the private key s_x used for signing the credential $cred_x$ would also invalidate, due to the forward secrecy, all signed credentials with an index greater than x. This approach is not applicable in our case for two reasons: First, the data subject wants to revoke certain consent without affecting previously given consent. Second, the processor wants to retain a verifiable proof for actions in the past.

5.2 Performance

Obviously adding a signature adds an overhead over unsigned data, both in size and in performance. The increase in size is due to the need to additionally store and transmit the following information: a digital signature and an optional number of substitution hashes.

The data itself, and each data item, can be of arbitrary size, while the digital signature is of fixed length. Each substitution hash is of fixed length, regardless of the data item's or sub-tree's size it represents. When existing X.509 certificates are reused to store the digital signature, accompanied with validity periods and other relevant information, this would result in adding approximately 1200 bytes, including the begin and end markers. Even when using 2048 bit RSA modulus for the keys the certificates are usually smaller than 2000 bytes. Optional, if data-items are omitted, the needed substitution hashes (for example using SHA-1) occupy additional 160 bits. Of course using longer hash functions (SHA-256 or SHA-512) slightly increases this overhead. In general, the omission of a single data item does not reduce the data set's size by the size of the omitted data item, as the substitution hash needs to be stored. On the other hand, if several data items are to be omitted, and they form a sub-tree of the hash tree, they can be substituted by just one single hash value.

We will now look at the overhead in performance, and when and where it occurs. Generally speaking two steps involve processing time: The hash tree and the asymmetric cryptography. We shortly show how expensive each of these operations are, then we will look where and how often this additional computing is needed to generate, process, and check the verifiable proof of consent.

The hash tree we used in our prototype is a complete binary tree. The number of leafs equals the number of data items rounded up to the next even number. So the number of hash operations needed to generate the tree is $\approx (2{*}\mathit{data\ items}){-}1$. Just to give you an indication, an Intel Core 2 Duo running at 2.4 GHz can roughly do 150 SHA-1 hashes on 1024 bits of data in 1 ms, and more on smaller data. Thus generating the root hash of a data set of 75 data items takes 1 ms. Since the original proposal by Merkle [14] in 1980, there have been several improvements on the generation and traversal of Merkle hash trees. In 2004 Szydlo [20] showed that it is possible to compute sequential tree leafs and authentication data in $2\log_2(leafs)$ time and $3\log_2(leafs)$ space.

Having the hash tree, the signature must be generated and verified. With RSA the signature generation is more labor intensive than the verification. The same Intel Core 2 Duo reported[4] about 30 signing and 1000 verify operations for 2048 bits per second. Having identified the performance overhead we will now look when and where it occurs.

The asymmetric distribution is helpful as the signature will be generated just once by the data subject during data dissemination. The data subject generates a single signature, once, when the consent is established. Additionally the data subject is not expected needing to submit high volumes of personal data in a short amount of time. On the other hand, the processors might need to process high volumes of personal data, but their operations are hash computations and signature verifications, both are less expensive.

Last, we look at the overhead required for querying the status provider. This results in communication overhead and one signature generation by the data subject or a service acting on his behalf and one signature verification by

[4] Using `openssl speed` of OpenSSL 0.9.8h

the processor that requested the status. Performance measures gathered from an unoptimized prototype that was implemented to secure the integrity and authenticity of fragmented web content [18] showed the following: Our Firefox extension needed less than one second to parse a website's DOM tree, identify data-items, generate the hash tree, query the OCSP responder, and verify the certificate's digital signature, including the verification of the certificate chain.

We suppose the increase in size is marginally compared to todays bandwidth and communication costs. The increase in size can be limited if larger or multiple data items are omitted. Here the tree based structure optimally allows to replace several omitted data items by just one intermediate node's substitution hash. The cryptographic operations introduce another, more important, overhead, but as we showed the more expensive operations are distributed among the data subjects. The bulk processing done by processors only involves less costly operations of hashing and signature verification.

6 Conclusion

Our solution allows services to gain a proof of consent even for aggregated personal data. Doing so without the need of a direct relationship with the data subject and without the iterative involvement of the data subject. Additionally our solutions caters for changes in the expression of consent, allowing to vanish a once established consent. Technically, our solution builds on digitally signed hash tree and reuses PKI mechanisms, especially certificates and certificate revocation. As Weitzner et al. we see the desirable properties in accountability and allowing the data subject to follow the use of his personal data. We balanced the accountability with the reassurance gained through the consent status check. This results in a win-win situation, the processor is reassured of that the data subject's consent is still established and the data subject is able to see this as an entry in his accountability log.

Privacy protection based on access control, thus restricting the flow of personal data upfront, shall be used whenever possible. However, ongoing trends like copy-left licenses, advances in semantic web, and increased digital footprints through the social web [12] show the need for a digital form of the data subject's initial expression of consent. Our work offers a provable expression of consent, in a form that allows passing it on, that can be retained, and that remains verifiable by third-parties. This was not available for aggregated personal data before.

Augmenting existing access control based privacy protection architectures, we offer accountability based architectures a proof of consent that the social and legal frameworks can rely upon. The already existing services that process data can easily add the proposed mechanisms, as trust and revocation mechanisms can be borrowed from existing PKI or web of trust systems. New services and business models can be built upon the new proof of consent.

Further research will look into the adoption of other suitable digital signature mechanism, and better fit aggregated data. We plan to integrate our approach into existing web services.

References

1. T. Berners-Lee. Semantic Web Road map. `www.w3.org/DesignIssues/Semantic.html`, Sept. 1998.
2. E. Bertino, B. Carminati, E. Ferrari, B. Thuraisingham, and A. Gupta. Selective and authentic third-party distribution of XML documents. *IEEE TKDE*, 16:1263–1278, 2004.
3. B. Carminati, E. Ferrari, and E. Bertino. Securing XML data in third-party distribution systems. In *Proceedings of 14th ACM CIKM*, pages 99–106, 2005.
4. P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. Stubblebine. Flexible authentication of XML documents. In *8th ACM Conf. on Computer and Comm. Security*, 2001.
5. EU. Directive 95/46/EC of the European Parliament and of the Council of 24 October 1995 on the protection of individuals with regard to the processing of personal data and on the free movement of such data, Oct. 1995.
6. Google. Google Friend Connect. `www.google.com/intl/en/press/annc/20080512_friend_connect.html`, May 2008.
7. R. Housley, W. Polk, W. Ford, and D. Solo. Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 3280 (Proposed Standard), April 2002. Updated by RFC 4325.
8. R. Housley, W. Polk, W. Ford, and D. Solo. RFC 3280 - internet X.509 PKI certificate and certificate revocation list (CRL) profile, Apr. 2002.
9. D. Hutter and M. Volkamer. Information flow control to secure dynamic web service composition. In J. A. Clark, R. F. Paige, F. Polack, and P. J. Brooke, editors, *SPC*, volume 3934 of *Lecture Notes in Computer Science*, pages 196–210. Springer, 2006.
10. Y.-L. Lai and K. L. Hui. Internet opt-in and opt-out: investigating the roles of frames, defaults and privacy concerns. In C. Shayo, K. Kaiser, and T. Ryan, editors, *CPR*, pages 253–263. ACM, 2006.
11. Z. Le, Y. Ouyang, Y. Xu, J. Ford, and F. Makedon. Preventing unofficial information propagation. In *ICICS*, pages 113–125, 2007.
12. M. Madden, S. Fox, A. Smith, and J. Vitak. PEW internet & american life project report: Digital footprints. `www.pewinternet.org/pdfs/PIP_Digital_Footprints.pdf`, Dec. 2007.
13. R. C. Merkle. *Secrecy, Authentication, and Public Key Systems,*. PhD thesis, Stanford, 1979.
14. R. C. Merkle. Protocols for public key cryptosystems. In *IEEE Symposium on Security and Privacy*, page 122, 1980.
15. J. L. Muñoz, J. Forné, and J. C. Castro. Evaluation of Certificate Revocation Policies: OCSP vs. Overissued-CRL. In *DEXA Workshops*, pages 511–518. IEEE Computer Society, 2002.
16. M. Myers, R. Ankney, A. Malpani, S. Galperin, and C. Adams. X.509 Internet Public Key Infrastructure Online Certificate Status Protocol - OCSP. RFC 2560 (Proposed Standard), June 1999.
17. T. O'Reilly. What is Web 2.0. `www.oreillynet.com/lpt/a/6228`, Sept. 2005.
18. H. C. Pöhls. ConCert: Content revocation using certificates. In *Sicherheit 2008*, volume 128 of *GI-Edition Lecture Notes in Informatics (LNI)*, pages 149–162, Saarbrücken, Germany, April 2008. GI.
19. A. C. Squicciarini, A. Bhargav-Spantzel, A. Czeskis, and E. Bertino. Traceable and automatic compliance of privacy policies in federated digital identity management.

In G. Danezis and P. Golle, editors, *Privacy Enhancing Technologies*, volume 4258 of *Lecture Notes in Computer Science*, pages 78–98. Springer, 2006.
20. Michael Szydlo. Merkle tree traversal in log space and time. In *Eurocrypt '04*, 2004.
21. W3C. The platform for privacy preferences 1.0 (P3P1.0) specification. `www.w3.org/TR/P3P/`, Apr. 2002.
22. D. J. Weitzner. Reciprocal Privacy (ReP) for the Social Web. `dig.csail.mit.edu/2007/12/rep.html`, Dec. 2007.
23. D. J. Weitzner, H. Abelson, T. Berners-Lee, J. Feigenbaum, J. Hendler, and G. J. Sussman. Information accountability. Technical Report MIT-CSAIL-TR-2007-034, MIT, Jun. 2007.