

VIGraph – A Framework for Verifiable Information

Anirban Basu^(✉), Mohammad Shahriar Rahman, Rui Xu,
Kazuhide Fukushima, and Shinsaku Kiyomoto

KDDI Research, Fujimino, Japan

{basu,mohammad,ru-xu,ka-fukushima,kiyomoto}@kddi-research.jp

Abstract. In order to avail of some service, a user may need to share with a service provider her personal chronological information, e.g., identity, financial record, health information and so on. In the context of financial organisations, a process often referred to as the *know your customer* (KYC) is carried out by financial organisations to collect information about their customers. Sharing this information with multiple service providers duplicates the data making it difficult to keep it up-to-date as well as verify. Furthermore, the user has limited to no control over the, mostly sensitive, data that is released to such organisations. In this *preliminary work*, we propose an efficient framework – Verifiable Information Graph or VIGraph – based on generalised hash trees, which can be used for verification of data with selective release of sensitive information. Throughout the paper, we use personal profile information as the running example to which our proposed framework is applied.

Keywords: Privacy · Hash tree · Verifiability · Selective disclosure

1 Introduction

Verifiable user identity information has always been a cornerstone in security and authentication. Most services, both online and offline, require a user to provide information to prove the user’s identity. Chronological non-identity related information is also used in other scenarios. For instance, in Japan, users may maintain their medicine prescriptions in a specific logbook that is checked before future consultations or prescriptions to avoid conflicting medications. Financial organisations use a formal process called the *know your customer* or KYC to obtain and maintain information about their customers. Traditionally, each organisational entity asking for such data has to obtain and verify the data independently, and keep copies of it for future references and legal obligations. The users, on the other hand, do not have a mechanism to selectively control the release of such sensitive information. A centralised or decentralised registry of such information

A. Basu is also a Visiting Research Fellow at the University of Sussex, UK and Rutgers University, USA.

helps with speeding up user identity verification, but it raises questions about privacy of the data in the hands of this third-party registry.

Our Contribution: In order to simplify the process for a service provider to collect, verify and maintain user information, a centralised (or distributed) solution works where the user is in charge of providing the information and a trusted third party maintains irreversible proofs of the components of such information. In this *position paper*, we propose a framework – the **Verifiable Information Graph** or the VIGraph – based on a generalised hash tree as a data structure. A hash tree enables verifiability of individual sub-trees without the full knowledge of the data contents. From the user’s perspective, this very property of the hash tree enables releasing, for verification, parts of the information for specific checks without compromising the privacy of the rest. For the sake of brevity, we focus on user profile data to describe our proposed framework. The framework can also be applied to other types of information with similar verifiability requirements; we leave this as an avenue for future work.

Paper Organisation: The rest of the paper is organised as follows. In Sect. 2, we propose the structure of the VIGraph and explain the operations on it. This is followed by a brief discussion and a description of the relevant state-of-the-art in Sect. 3, before concluding with future directions in Sect. 4.

2 VIGraph: Signed Hash Tree with Optional Dependency Overlay

Hash trees, and the specialised Merkle Trees [1] have a specific property: each sub-tree of the tree can be verified independently of the other. This facilitates obfuscation on the actual contents of the data in sub-trees, which the verifier is not interested in. Our proposed data structure, the VIGraph, is four-levels deep, as illustrated in Fig. 1. Starting at the top level (root), the actual data is on the fourth level down in the leaf nodes. VIGraph is based on a generalised hash tree, which means that a non-leaf node can have more than two children, as opposed to a binary hash tree. The structure of the VIGraph closely resembles a tree but is technically a graph due to the optional dependency overlay. However, throughout the paper, we will use terminology akin to trees, such as leaf nodes and root. The purpose of the VIGraph is verifiability and selective information release; and not a fast search from the root. Thus, VIGraph is not to be confused or compared with hash tries or hash array mapped tries (HAMT).

The entities involved with the VIGraph are: (a) the user u whose information is being maintained, (b) organisational entities (each represented as o) responsible for endorsing (through digital signatures) various components of the user’s information, e.g., the driving license identifier of a user may be signed by the driving license issuing authority, (c) the metadata provider organisation(s) (each represented as p) responsible for maintaining and facilitating verification using

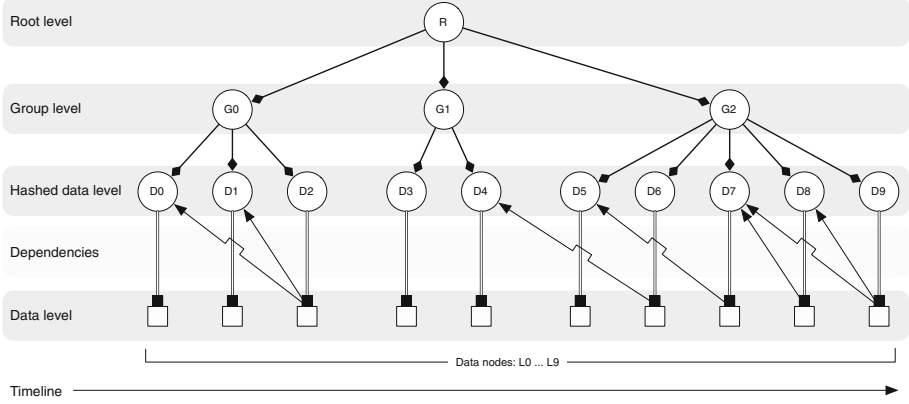


Fig. 1. Structure of a Verifiable Information Tree (VIGraph). The dependencies are optional. One-to-one, one-to-many, many-to-one and cross-group dependencies are illustrated.

the VIGraph; and (d) the service provider or a verifier organisation (represented as v) responsible for verifying information against that stored in the VIGraph.

In short, each user has her VIGraph representing information about her, parts of which are maintained by one or more metadata providers; and the information maintained by the graph may or may not be endorsed by relevant information issuing authorities.

2.1 Levels and Dependencies

Figure 2 illustrate the contents of the nodes at various levels of the VIGraph.

Data Level: The leaf nodes $L_0 \dots L_9$ (refer to Fig. 1) are the actual data, L_i , such as (using the personal profile example) national identity information, residential addresses, driving license and so on. Each node may also contain optional dependency pointers, described later.

Hashed Data Level: The hashed data nodes, $D_0 \dots D_9$, on the *hashed data level*, contain the hashes of the data node contents, such that $D_i = h(L_i, r_u)$ where $h(L_i, r_u)$ is some cryptographically secure one-way hash function that computes the hash on data L_i along with some random number r_u . The random value, r_u is different for different users. This ensures that the hash of the same information, e.g., a certain residential address is different for different users, thus thwarting linkability of information if it belongs to two different users. In addition to the hashes, each hashed data node also contains up to two signatures for the corresponding hash value. The mandatory signature is from the user, $sig_u(h(L_i, r_u))$, while the other optional signature is from the respective organisation, $sig_o(h(L_i, r_u))$, in charge of issuing the data as shown in Fig. 2.

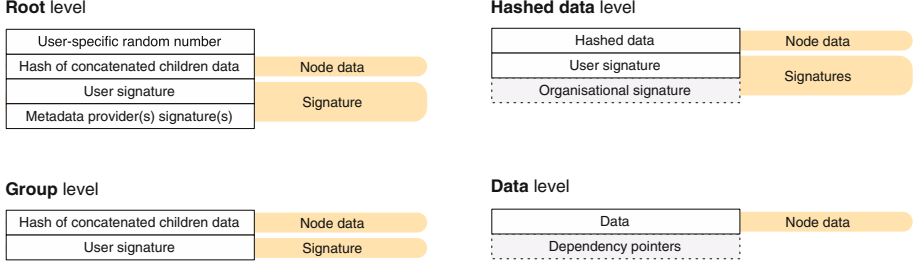


Fig. 2. Contents of the nodes at various levels of the VIGraph.

For example, in Japan, if L5 denotes the user’s current residential address then D5 will contain the hash of L5, the signature from the user and the signature from the respective city or ward office endorsing the information.

Group Level: In the *group level*, the group nodes are defined by the user and are useful for semantic grouping as well as privacy during verification. For instance, national identity information may be in a group with visa information while residential addresses could be in a different group. Each group contains the hash of the concatenation of the hash values (of the actual data) stored in its children data nodes. The newest child in the group appears rightmost in the concatenation. Each group also contains the user’s signature of its contents.

Root Level: The *root level* node contains a similar concatenation of its children group nodes. The root node also contains the user-specific random number, r_u , a signature from the user and another from the metadata provider maintaining the graph.

Dependency Overlay: The dependencies can exist between the data level nodes and the hashed data level nodes, with one-way pointers from the data level to the hashed data level. Dependencies can help retrieve related data, are defined when adding new data and are immutable after that. For example, if a user holding the nationality of country C_1 acquires a work-permit in country C_2 to live and work there, then the data node for the work permit may have an optional pointer to the signed data for the citizenship. This signifies the semantic dependency that the visas depend on a specific nationality. While declaring the optional dependency is the user’s choice, the signing authority, o , for the corresponding hashed data level node may reject a certain declared dependency if it is deemed irrelevant, or conversely require the declaration of a specific dependency. The dependencies can be one-to-one, one-to-many, many-to-one, many-to-many and cross-group. All dependencies are backward links, in terms of time.

2.2 Storage of the VIGraph

Parts of the VIGraph will be stored by the user as well as the metadata provider(s). There can be more than one metadata provider, in which case the VIGraph may be stored across multiple metadata providers with erasure coding to ensure reliability. The user will store everything but the root whereas the metadata provider will store everything but any node from the data level – thus, the user is responsible for storing the actual data. This allows the graph to be verified in part or full against the version stored by the metadata provider(s) without having them to also maintain a copy of the actual data, which can contain sensitive information.

2.3 Data Operations on the VIGraph

Add Operation: The add operation enables adding leaf nodes at the data level to an empty graph or an existing one. In both cases, we assume that the user has already generated a public key, private key pair.

Figure 3(a) illustrates the steps the user needs to take to add a new data node to an empty, i.e., non-existent graph. The states: *Obtain organisational signature* and *Publication* depend on validation from the organisational signatory, o and the metadata provider, p , respectively. The entity o may not provide the optional signature. Figure 3(b) illustrates the steps the user needs to take to add a new data node to an existing graph. Depending on whether the new data is being added to a new group or an existing group, the add operation will require the validation, by the metadata provider, of the optional group hash update and the root hash update. The user must prove to the metadata provider the knowledge of the existing group and root hashes, and the fact that those existing hashes indeed update

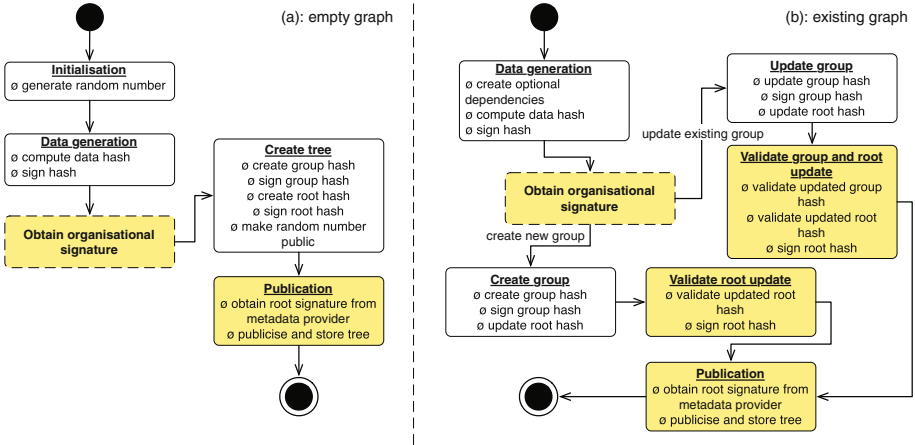


Fig. 3. The user-side state diagrams for the add data operation to: (a) an empty graph, and (b) an existing graph.

to the new ones when the new data is added. In both cases of adding a node to a new graph or an existing graph, the metadata provider may not sign the root hash of the graph if it fails to check, including offline mechanisms beyond the scope of this paper, the validity of the information being added.

Delete Operation: The user can only ‘remove’ a data leaf node from local storage. This does not delete the corresponding hashed data node but it will ensure that the original data node is no longer recoverable. The hashed data node can still be used in verification of other data nodes without compromising the privacy of the actual data that it encapsulates. Removing the organisational signature during the verification process further ensures that the verifier is unable to guess what the original data deleted node was, given its hashed value only.

Group Restructuring: Group structuring allows one data leaf node and its corresponding hashed data node to be moved from one group to another. The move operation is effectively a delete operation followed by an add operation. Since the node being moved is not removed from the graph, moving groups may result in inter-group dependencies. Suppose D_x denotes the concatenated hashes of all the other sibling data hash nodes (in the hashed data level), i.e., the concatenations of some $D_x = h(L_i, r_u) || \dots || h(L_{k-1}, r_u)$. To delete an existing node L_k from group G_k , the user needs to prove, to the metadata provider, that the hash contained in the existing G_k is $h(D_x || h(L_k, r_u))$. Once validated, the hash contained in G_k that has been updated to $h(D_x)$ is signed by the user. The rest of the move operation is just the same as an add operation to an existing graph, except that $h(L_k, r_u)$ will already have the necessary signatures on it.

2.4 Data Verification

The verification process is illustrated in Fig. 4. The verifier, e.g., a service provider, requests two sets of data; one from the user and the other from the metadata provider(s). Note that only the signed root hash and the user-specific random number are the required information from the metadata provider and the

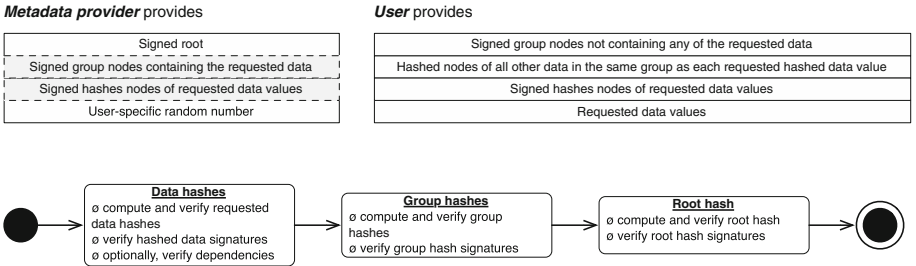


Fig. 4. The required information and the process for verification.

rest are optional. The verification process has three stages. Firstly, the verifier computes the hashes on the data provided by the user and cross-checks if those hashes match the ones obtained from the user and the metadata provider(s), and that their signatures are correct. The verifier can also check the optional dependencies at this stage. Secondly, the verifier uses information of all other provided data hashes to compute group hashes, and check if these and their signatures match those obtained from the user and the metadata provider. Finally, it computes and verifies the root hash in the same way and checks against the one obtained from the metadata provider(s). It is evident that in order to verify an information component (in the data level), it is necessary for the user to provide information of the hashed data level of all the other children of the group. The larger the number of children, the more inefficient this is. This shows, as indicated before, that having one or few groups to represent all data is inefficient although the framework will still work.

3 Discussion and the State-of-the-art

Blockchains [2,3] have been used [4] to store anonymised but verifiable information for identity and access control, including commercial work from KPMG¹ and Deloitte². Blockchains are considered as the so-called ‘trustless’ systems that remove the need to trust any entity in particular because a decentralised consensus mechanism provides majority opinion. The idea of a trustless system is questionable because the verifier has to trust the accuracy of the consensus mechanism and the underlying hash function instead of any specific entities. With a public blockchain involving a very large number of participants, the probability of collusion and hence alteration of majority opinion is low but not impossible. With a private blockchain involving a limited number of participants, the risk of collusion is higher. Furthermore, public blockchains are not scalable since all participants have to maintain a very large hash chain, and agree on a consensus. In addition, maintaining the information of every user in a single blockchain may prove wasteful in certain situations. While the fault tolerance of blockchains is often attributed to their decentralised nature, it is to be noted that other decentralised means of storage of information, e.g., erasure coding, also achieves fault tolerance without the downsides of the consensus algorithm. In our work, we show that hash trees alone can be sufficient for verifiability of selectively released information.

Selective release of personal information is a well-studied subject. Kiyomoto et. al’s work on privacy policy manager [5] discusses a framework that enables interpreting privacy policies easier for users, which has been standardised by the oneM2M initiative [6]. Sanitizable signatures schemes [7–9] allow a semi-trusted third party called sanitizer to sign on a modified message using just the public

¹ See: <https://goo.gl/oaECro>.

² See: <http://www.deloitte.co.uk/smartid/> and <https://github.com/SmartIdentity/smartId-contracts>.

key from the signer without interacting with her. The constraint is that the modifications lie in the predetermined parts of the original message by the signer. This kind of signatures is useful when the message owner wants to release multiple versions of a message. It is applicable in the situation where some sensitive information of the message need anonymisation. However, it can not be directly applied to the case of verification with selective release, which is the concern of our work. Interestingly, sanitizable signatures can be viewed as a work orthogonal to ours. As a future direction, we may utilise it in our VIGraph proposal to provide better privacy protection. Aggregate signatures [10] can also be utilised to tackle the requirements of KYC since it allows the addition of signatures. The signatures of n messages from n different signers can be aggregated into a single signature, which can verify the integrity of all the n messages. Attribute-based access control [11] can provide fine-grained access credentials to different entities, but is different from verifiability of fine-grained information. Leung and Mitchell [12] proposed a privacy preserving authentication protocol, which allows a service provider to successfully identify the user without breaching the user's private information. Direct Anonymous Attestation (DAA) schemes are special signatures which provide a balance between signer authentication and privacy.

4 Conclusion and Future Work

In this paper, we have presented a generalised hash tree based framework for verifying selectively-released information through a trusted third party, which stores different parts of the hash graph but not the actual data. We have discussed how our framework enables selective release of data for verification, thus helping with privacy. Throughout this paper, we have used personal profile data as an example but our framework can be applied to other types of data. Beyond what we have identified as future work in the paper, we aim to refine the deletion operation in our framework to make it compliant with right-to-forget. We also plan to implement a prototype, and run user and performance evaluations; and compare our proposal more extensively with blockchain based alternatives.

References

1. Merkle, R.C.: Method of providing digital signatures (1979). <https://www.google.com/patents/US4309569>
2. Nakamoto, S.: Bitcoin: a peer-to-peer electronic cash system (2008)
3. Swan, M.: Blockchain: Blueprint for a New Economy. O'Reilly Media, Inc., Sebastopol (2015)
4. Zyskind, G., Nathan, O., et al.: Decentralizing privacy: using blockchain to protect personal data. In: Security and Privacy Workshops (SPW), pp. 180–184. IEEE (2015)
5. Kiyomoto, S., Nakamura, T., Takasaki, H., Watanabe, R., Miyake, Y.: PPM: privacy policy manager for personalized services. In: Cuzzocrea, A., Kittl, C., Simos, D.E., Weippl, E., Xu, L. (eds.) CD-ARES 2013. LNCS, vol. 8128, pp. 377–392. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-40588-4_26](https://doi.org/10.1007/978-3-642-40588-4_26)

6. Datta, S.K., Gyrard, A., Bonnet, C., Boudaoud, K.: oneM2M architecture based user centric IoT application development. In: 2015 3rd International Conference on Future Internet of Things and Cloud (FiCloud), pp. 100–107. IEEE (2015)
7. Ateniese, G., Chou, D.H., Medeiros, B., Tsudik, G.: Sanitizable signatures. In: Vimercati, S.C., Syverson, P., Gollmann, D. (eds.) ESORICS 2005. LNCS, vol. 3679, pp. 159–177. Springer, Heidelberg (2005). doi:[10.1007/11555827_10](https://doi.org/10.1007/11555827_10)
8. Miyazaki, K., Hanaoka, G., Hideki, I.: Invisibly sanitizable digital signature scheme. IEICE Trans. Fundam. Electron. Commun. Comput. Sci. **91**(1), 392–402 (2008)
9. Brzuska, C., Fischlin, M., Freudenreich, T., Lehmann, A., Page, M., Schelbert, J., Schröder, D., Volk, F.: Security of sanitizable signatures revisited. In: Jarecki, S., Tsudik, G. (eds.) PKC 2009. LNCS, vol. 5443, pp. 317–336. Springer, Heidelberg (2009). doi:[10.1007/978-3-642-00468-1_18](https://doi.org/10.1007/978-3-642-00468-1_18)
10. Boneh, D., Gentry, C., Lynn, B., Shacham, H.: Aggregate and verifiably encrypted signatures from bilinear maps. In: Biham, E. (ed.) EUROCRYPT 2003. LNCS, vol. 2656, pp. 416–432. Springer, Heidelberg (2003). doi:[10.1007/3-540-39200-9_26](https://doi.org/10.1007/3-540-39200-9_26)
11. Goyal, V., Pandey, O., Sahai, A., Waters, B.: Attribute-based encryption for fine-grained access control of encrypted data. In: Proceedings of the 13th ACM Conference on Computer and Communications Security, pp. 89–98. ACM (2006)
12. Leung, A., Mitchell, C.J.: Ninja: non identity based, privacy preserving authentication for ubiquitous environments. In: Krumm, J., Abowd, G.D., Seneviratne, A., Strang, T. (eds.) UbiComp 2007. LNCS, vol. 4717, pp. 73–90. Springer, Heidelberg (2007). doi:[10.1007/978-3-540-74853-3_5](https://doi.org/10.1007/978-3-540-74853-3_5)

Trust Management XI

11th IFIP WG 11.11 International Conference, IFIPTM

2017, Gothenburg, Sweden, June 12-16, 2017,

Proceedings

Steghöfer, J.-P.; Esfandiari, B. (Eds.)

2017, X, 229 p. 32 illus., Hardcover

ISBN: 978-3-319-59170-4