A system and method to offer confidentiality of transactions in Blockchain-based systems

Angelo De Caro, Elli Androulaki, Alessandro Sorniotti,
Thorsten Kramp, Binh Nguyen, Ramesh Gopinath

1. Background

A Blockchain system (B) consisting of clients (Cs) (submitting transactions), and validating entities (Vs) (executing and validating transactions). Transactions submitted by clients are signed with client certificates managed by an identity management infrastructure (IDM). A chaincode consists of one or more functions that can be executed in B. There are at least two types of transactions in the system:

* Deploy Transactions, which are used to deploy chaincode to B;
* Invoke Transactions, which are used to invoke a chaincode.

A transaction consists of, at least, a Chaincode ID (CID), that identifies the chaincode, and a Payload (P) whose content changes depending on the type of the transaction.

Here we also assume that there is a consensus protocol in place such that transactions are totally ordered, or ordered in blocks, before they are executed/validated.

Through this invention we aim to deal with the fact that some of the transactions that a client can submit can have confidentiality constraints w.r.t. all parties who don't belong to B. This offers properties such as Blockchain-data-at-rest protection, as well as Blockchain-data-in-flight protection.

This method offers a simple and efficient way to enforce confidentiality in the setting described above and is agnostic to the underlying consensus algorithm*. Moreover, fine-grained auditing capabilities are provided.

2. Related Work

Two previously proposed systems relate to his work, to our knowledge:

- Hawk[1]. Hawk is a framework for building privacy-preserving smart contracts in decentralized cryptocurrency systems with the following security guarantees: (1) On-chain privacy that stipulates that transactional privacy be provided against the public (i.e., against any party not involved in the contract) – unless the contractual parties themselves voluntarily disclose information, (2) contractual security protects parties in the same contractual agreement from each other. In contrast our technique is designed to offer transactions confidentiality in permissioned networks with strong users identification and without an embedded cryptocurrency.
- Enigma [2]. Enigma is another project where confidential contract execution is dealt with in untrusted environment. In particular, similar to Bitcoin, Enigma removes the need for a trusted third party, enabling autonomous control of personal data. Computations and data storage are not replicated by every node in the network, but only a small subset perform each computation over different parts of the data. However, their computational model is based on secure multi-party computation, that is not practical, and requires all parties to be online and exchange messages for a contract to be executed.

3. Summary of our method

Confidentiality is a crucial requirement to allow parties having common interests to exchange information in a protected way w.r.t any third party who is not part of the system and has incentives to eavesdrop B in order to extract confidential information.

We stay simple and efficient, and employ only symmetric key encryption. In this setting, one of the main threats is represented by cryptanalysis[3,4] that leverage the availability of many ciphertexts being encrypted under the same key. Another important challenge that is specific to the Blockchain setting, is that validators need to run consensus over the state of the Blockchain, that, aside the transactions themselves, also includes the state updates of individual contracts or chaincodes. Though this is trivial to do for non-confidential chaincodes, for confidential chaincodes, one needs to design the state encryption mechanism such that the resulting ciphertexts are semantically secure, and yet, identical if the plaintext state is the same.

To overcome both challenges, we design a key hierarchy that reduces the number of ciphertexts exposed, that are encrypted under the same key. At the same time, as we use some of these keys for the generation of IVs, this allows the validating parties to generate exactly the same ciphertext when executing the same transaction (this is necessary to remain agnostic to the underlying consensus algorithm) and offers the possibility of fine-grained auditing by disclosing to auditing entities only the most relevant keys.

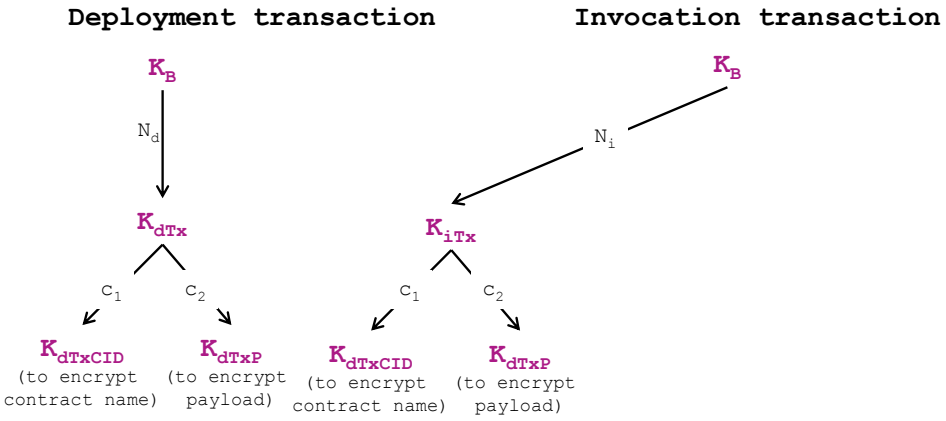
4. Detailed description of our method

We assume that (though we do not deal with the details of how this can be achieved) when B is first deployed, IDM generates a symmetric key for B (K_B) that is distributed at registration time to all the entities of B, i.e., the clients and the validating entities of B. As we see later, our method requires the enclosure in a transaction of a random number called nonce. This would also guarantee that two transactions that are supposed to issue the same end-user operation cannot be identical.

In order to defeat crypto-analysis and enforce confidentiality, we envision the following key hierarchy, and generation and validation of confidential transactions:

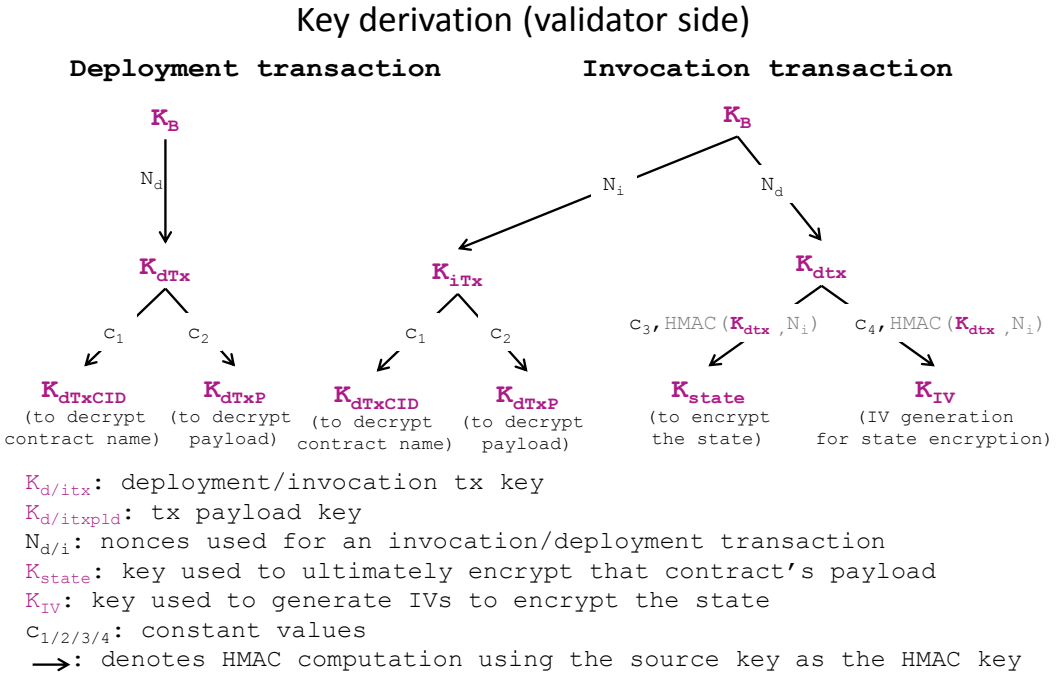
To submit a Confidential Transaction (Tx) to B, a client first samples a nonce (N), which is required to be unique among all the transactions submitted to B, and derive a transaction symmetric key (K_{Tx}) by applying the HMAC function keyed with K_B and on input the nonce, K_{Tx}= HMAC(K_B, N). From K_{Tx}, the client derives two AES keys (K_{TxCID} as HMAC(K_{Tx}, c1), K_{TxP} as HMAC(K_{Tx}, c2)) to encrypt respectively CID and P. c1, c2 are public constants. The nonce, the Encrypted Chaincode ID (ECID) and the Encrypted Payload (EP) are added in the transaction Tx structure, that is finally signed and so authenticated. Figure below shows how encryption keys for the client's transaction are generated. Arrows in this figure denote application of an HMAC, keyed by the key at the source of the arrow and using the number in the arrow as argument. Deployment/Invocation transactions' keys are indicated by d/i respectively.

Key derivation (client side)



$K_{d/itx}$: deployment/invocation tx key
$K_{d/itxpld}$: tx payload key
$N_{d/i}$: nonces used for an invocation/deployment transaction
$c_{1/2}$: constant values
$\longrightarrow$: denotes HMAC computation using the source key as the HMAC key

To validate a confidential transaction Tx submitted to B by a client, a validating entity first decrypts ECID and EP by re-deriving K_{TxCID} and K_{TxP} from K_B and Tx.Nonce as done before. Once the Chaincode ID and the Payload are recovered the transaction can be processed.



$K_{d/itx}$: deployment/invocation tx key
$K_{d/itxpld}$: tx payload key
$N_{d/i}$: nonces used for an invocation/deployment transaction
K_{state}: key used to ultimately encrypt that contract's payload
K_{IV}: key used to generate IVs to encrypt the state
$c_{1/2/3/4}$: constant values
$\longrightarrow$: denotes HMAC computation using the source key as the HMAC key

When V validates a confidential transaction, the corresponding chaincode can access and modify the chaincode's state. V keeps the chaincode's state encrypted. In order to do so,
V generate symmetric keys as depicted in the figure above. Let iTx be a confidential transaction invoking a function deployed at an early stage by the confidential transaction dTx (notice that iTx can be dTx itself in the case, for example, that dTx has a setup function that initializes the chaincode's state). Then, V generates two symmetric keys K_{IV} and K_{state} as follows:

1. It computes as K_{dTx} , i.e., the transaction key of the corresponding deployment transaction, and then N_{state} = HMAC(K_{dtx} ,hash(N_i)), where N_i is the nonce appearing in the invocation transaction, and "hash" a hash function.
2. It sets K_{state} = HMAC(K_{dTx}, c_3 || N_{state}), truncated opportunely deeding on the underlying cipher used to encrypt; c_3 is a constant number
3. It sets K_{IV} = HMAC(K_{dTx}, c_4 || N_{state}); c_4 is a constant number

In order to encrypt a state variable S, V first generates the IV as HMAC(K_{IV}, crt_{state}) properly truncated, where crt_{state} is a counter value that increases each time a state update is requested for the same chaincode invocation. The counter is discarded after the execution of the chaincode terminates. After IV has been generated, V encrypts with authentication (i.e., GSM mode) the value of S concatenated with N_{state}(Actually, N_{state} doesn't need to be encrypted but only authenticated). To the resulting ciphertext (CT), N_{state} and the IV used is appended. In order to decrypt an encrypted state CT|| $N_{state'}$, V first generates the symmetric keys K_{dTX} ',K_{state} 'using $N_{state'}$ and as and then decrypts CT.

Generation of IVs: In order to be agnostic to any underlying consensus algorithm, all the validating parties need a method to produce the same exact ciphertexts. In order to do use, the validators need to use the same IVs. Reusing the same IV with the same symmetric key completely breaks the security of the underlying cipher. Therefore, the process described before is followed. In particular, V first derives an IV generation key $K_{IV}\_$ by computing HMAC($K\_{dTX,}$ c_4 || N_{state}), where c_4

is a constant number, and keeps a counter $crt_{state}$for the pair (dTx, iTx) with is initially set to 0.. Then, each time a new ciphertext has to be generated, V generated a new IV by computing it as the output of HMAC(K_{IV}, crt_{state}) and then V increments the crt_{state} by one.

Another benefit that comes with the above key hierarchy is the ability to allow fine-grained auditing. For example, by releasing K_B we give access to the whole chain, by releasing only K_{state}dTx_iTx for a given pair of transactions (dTx,iTx) we give access to state updated by iTx, and so on.

References:

[1] Kosba et.al. Hawk: The Blockchain Model of Cryptography and Privacy-Preserving Smart Contracts., ePrint https://eprint.iacr.org/2015/675.pdf, 2015

[2] G. Zyskind, O. Nathan, A. Pentland. Enigma. Decentralized Computation Platform with Guaranteed Privacy http://enigma.media.mit.edu/enigma_full.pdf, 2015

[3] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full AES. ASIACRYPT 2011, 2011

[4] Alex Biryukov, Dmitry Khovratovich, Ivica Nikolic. Distinguisher and Related-Key Attack on the Full AES-256. CRYPTO 2009