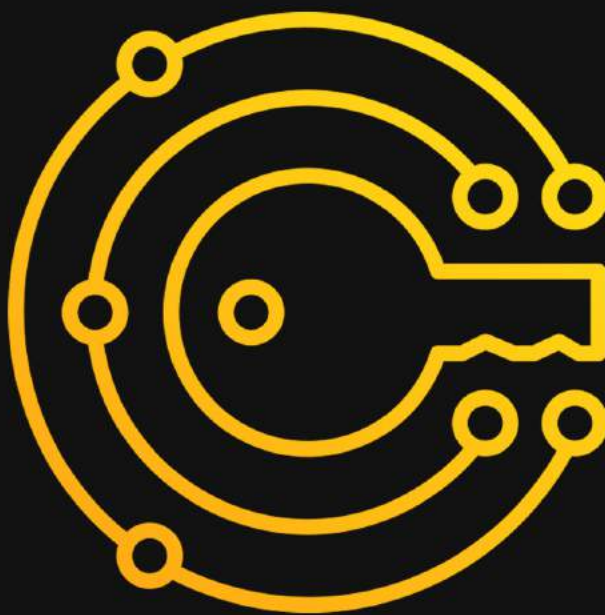


EDITED 4.24.2018

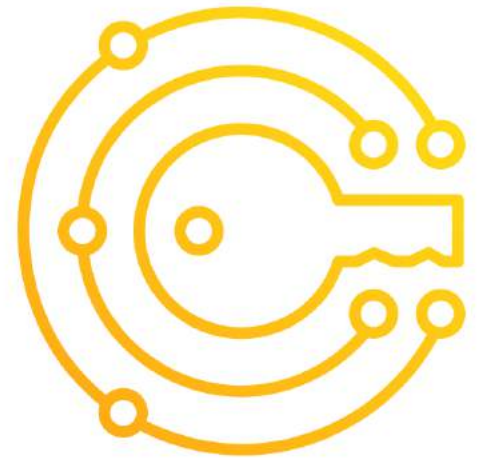


WHITEPAPER

TECHNICAL VERSION



WHITEPAPER
TECHNICAL OVERVIEW
<https://crypticcoin.io/>



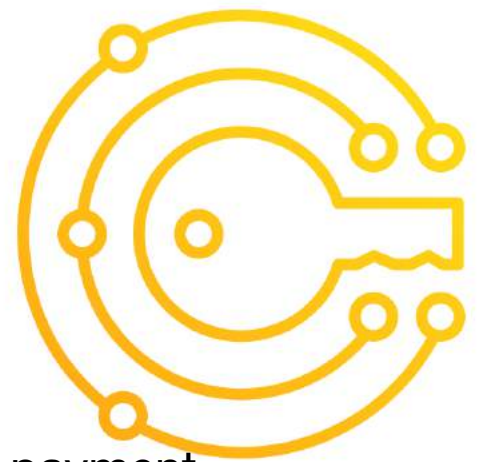
Forward

All technologies have both advantages and disadvantages. That's why we are combining technologies into the creation of one coin. CrypticCoin gives our users the ability to choose, compare and eventually use the confidentiality mechanism that best suits them. CrypticCoin is a coin designed for the discretion of users by offering them the appropriate tools.

As a basis, we decided to implement a mix of an enriched version of the ZeroCash Protocol (zk-SNARK systems protocol) and a Hybrid version of Verge (Stealth Addressing Technologies). However, we make no claims that CrypticCoin will offer only these two mechanisms, over time, other privacy technologies (e.g., Confidential Transactions) will be implemented in CrypticCoin. Over time other privacy technologies will be added to CrypticCoin. CrypticCoin also has proprietary enhanced security measures built in to further enhance the system of security within the CrypticCoin Ecosystem.

It is with such functionality that CrypticCoin will be presented to the community. We suggest and encourage users have shielded transactions enabled by default.





Introduction

Most cryptocurrencies are based on blockchains in which payment transactions are stored "as is" in a decentralized ledger. Because the blockchain is public, said details such as a sender's public address, recipients public address, and payment amount about each transaction as well as the history of all transactions can be viewed by anyone. While public addresses are not explicitly tied to users' real identities, there are ways to learn more about users, their spending habits and relationships with each other using information stored in most blockchains. In most cases, wallets used for making transactions do not support anonymity features while connecting to blockchain nodes. A user's location can be determined by IP address of the used device and privacy of the transaction is eliminated.

CrypticCoin (CRYP) is a decentralized and open-source cryptocurrency that aims to connect best practices regarding the privacy and anonymity for its users. CrypticCoin allows users to engage in direct transactions rapidly and a high level of privacy.





MAX SUPPLY DEFINED

As of 1-18-2018 there were roughly **7,598,607,351** people exist on Planet Earth, therefore maximum supply of 7,598,607351 coins will be distributed. **One for every person on Earth.** We all deserve to control and manage our own privacy!

Security, Anonymity & Privacy All In One!



WHITEPAPER
TECHNICAL OVERVIEW
<https://crypticcoin.io/>

Enriched ZeroCash Protocol Integration



CrypticCoin further enhances levels of security and privacy by using the Zerocash protocol to its advantage. Utilizing the ZeroCash Protocol improves privacy by adding levels of transactions and ensures that payment transactions do not contain any public information about the sender's address, recipient's address or any transferred amounts. This is achieved by adding a sub-coins level to the existing base-coins. Each user can convert base-coins into sub-coins (1:1) to be able to make more private payment transactions. Users can also convert sub-coins back into base-coins (1:1) at any time when they want.

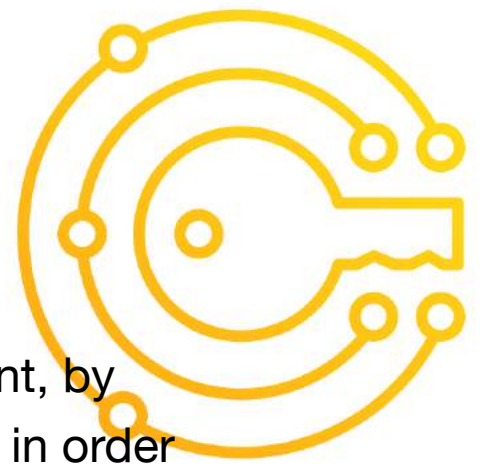
CrypticCoin further utilizes the enriched Zerocash functionality by implementing two new types of transactions: Fresh Mint transactions and Pour transactions, which are recorded to the public ledger as well.

A fresh mint transaction allows users to convert a specified number of base-coins (debited from any one of the owned base-coins accounts) into the same number of sub-coins (credited to any one of the owned sub-coins accounts).

The fresh mint transaction itself consists of a cryptographic commitment, which specifies the amount of converted sub-coins, owner address and unique serial number. The commitment is based on the SHA-256 hash function, that allows to hide both the converted amount and owner address. However, the commitment is constructed so that anyone can verify that the committed sub-coin has the claimed value.



Enriched ZeroCash Protocol Integration



A pour transaction allows a user to make a private payment, by consuming some amount of sub-coins owned by the user in order to produce the same amount of sub-coins to the recipient. The correctness of the transaction is validated via the use of zero-knowledge proof

(https://en.wikipedia.org/wiki/Zero-knowledge_proof).

A pour transaction, for (up to) two input sub-coins and (up to) two output sub-coins, involves proving, in zero knowledge, that:

- The user owns the two input sub-coins.
- Each one of the input sub-coins appears in some previous mint transaction or as the output sub-coin of some previous pour transaction.
- The total value of the input sub-coins equals the total value of the output sub-coins.

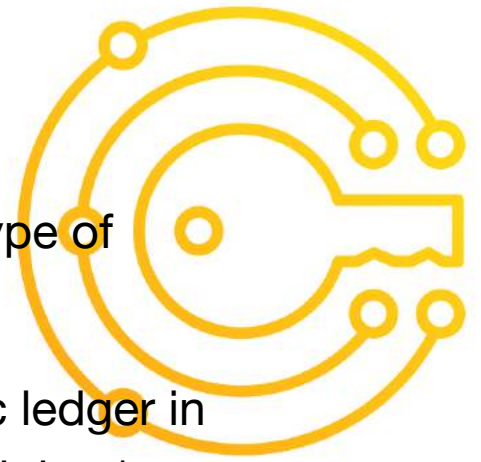
The pour transaction consumes the input sub-coins by revealing their serial numbers, but does not reveal any other information such as the amount of the input or output sub-coins, or the addresses of their owners.

The pour transaction can also output some amount of base-coins. This feature can be used to convert sub-coins back into base-coins or to pay transaction fees.

For a pour transaction, anyone can verify that the zero-knowledge proof contained therein is valid. For efficiency, our integrated use of the Zerocash Protocol uses "Zero-Knowledge Succinct Non-interactive Arguments of Knowledge" (zk-SNARK) systems, which are zero-knowledge proofs that are particularly short and easy to verify.



Privacy on/off options for transactions



With CrypticCoin users have the ability to choose what type of transaction they want to make - public or private.

A public transaction is recorded to the CrypticCoin public ledger in an unchanged form with the sender's public address, recipient's public CrypticCoin address and payment amount of CRYPT transferred.

This type of transaction can be done if it is necessary to prove to a third party that the sender did, in fact, make a particular transaction.

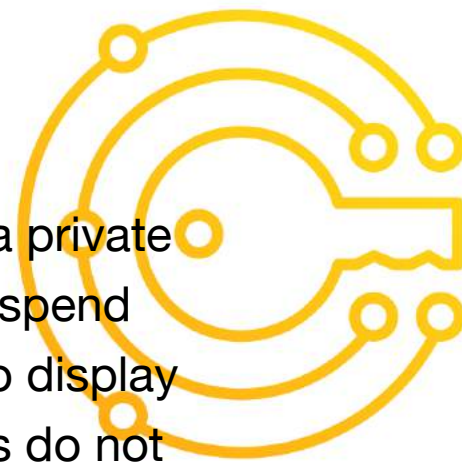
A private CrypticCoin transaction is recorded to the public ledger using a one-time public key which is generated by a special algorithm. When analyzing blockchain, such transactions are not traceable on a public ledger and can not be uniquely mapped to another transaction, so there is no possibility to analyze the activity of any network member using the public ledger, with whom and in what quantity exchanged in the transactions.

Private transactions will be improved by the implementation of the ZeroCash protocol, which completely hides the transferred amount and the transaction metadata. It will increase the privacy of payments by consuming coins transferred by a sender in order to produce the same amount of new coins for a recipient. So, received coins will not have a history. Regardless of a transaction type, the connection between a user's wallet and the blockchain are anonymized by default through hiding the user's real IP address when making transactions.

High levels of privacy and anonymity are achieved by the effective integration of such technologies as: Hybrid Stealth Addressing, ZeroCash protocol and IP Obfuscation.



Hybrid Stealth Addressing



When users create a CrypticCoin account they will have a private view key, a private spend key, and a public address. The spend key is used to send payments and the view key is used to display incoming transactions destined for users accounts. Users do not need to interact with these keys directly, all accounts (and their corresponding keys) are managed by a user's wallet. Nevertheless, the owner of the wallet can access them if necessary. Both the spend key and view key are used to build your CrypticCoin public address, which users present to a sender for receiving payments. Users can increase their privacy by providing different CrypticCoin public addresses for senders (users create separate accounts for interactions with different senders). But even in this case, all user transactions with a particular sender are linked with each other in public ledger using the same CrypticCoin public address.

We have enriched the Stealth Addressing technology which allows users to publish one address for everyone (a Hybrid Stealth Address) and at the same time greatly increases the privacy of received payments.

Hybrid Stealth Addresses allow/require the sender to create random one-time CrypticCoin public addresses for every transaction on behalf of the recipient. So, the recipient will have all of their incoming payments go to unique CrypticCoin public addresses on the blockchain, where they cannot be linked back to either the recipient's published addresses (stealth or public) or any other transactions addresses.

These unique CrypticCoin public addresses can only be recovered and spent by the recipient. By using Hybrid Stealth Addresses, only the sender and receiver can determine where a payment was sent. No one else will have the ability to link the wallet addresses together by investigating transactions on the blockchain.



Hybrid Stealth Addressing

Hybrid Stealth Addressing key features:

- Hybrid Stealth Addresses cannot be linked publicly to either the randomly created one-time CrypticCoin public address or any other one-time CrypticCoin public addresses.
- Hybrid Stealth Addresses can be recovered and spent by the recipient only.
- Only the recipient can link together all the payments made using their Hybrid Stealth Addresses.

Hybrid Stealth Addressing functionality is achieved through the Elliptic Curve Diffie-Hellman (ECDH) cryptography system. CrypticCoin Hybrid Stealth Address is a string that consists of a public view key and the public send key of the recipient. While making a payment, any sender is able to calculate a unique one-time public key for the recipient's new output on basis of this Hybrid Stealth Address. ECDH algorithm ensures that the one-time public key cannot be reverse engineered and that nobody else can duplicate it. This output can be located by the recipient's wallet during scanning the blockchain with wallet's private view key. After the output is detected and retrieved, recipient's wallet can calculate a one-time private key that corresponds with the one-time public key of the output. So, the recipient's wallet always knows about the current available balance. The recipient can spend the relevant output with their wallet's private spend key.



IP Obfuscation



To improve the privacy of connectivity between a user's wallet and the blockchain, all wallet distros (full and lightweight) will support anonymity features out of the box. This is done via integration of IP obfuscation service into distros.

Tor ("The Onion Router") is used as an IP obfuscation service. It is a decentralized system that enables anonymous communications through a network of relays which serve to obfuscate IP addressing information by bouncing connections from node to node at random, effectively eliminating any information trails.

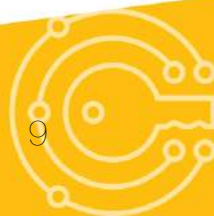
Tor redirects the users internet traffic through a free worldwide relay network to hide users location and interests from anyone who monitors networks or makes traffic analysis.

Tor uses the onion routing mechanism, which is implemented by nested encryption on the application layer of TCP/IP stack.

Tor encrypts transmitted data, including the next node destination IP, and sends it through a virtual circuit of randomly selected relays.

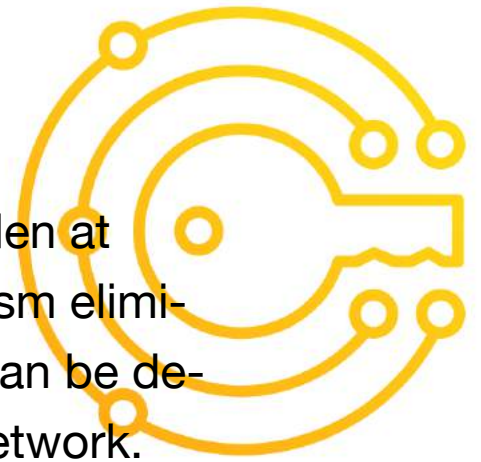
Each relay in a virtual circuit decrypts only the necessary layers of the data packets to find out which relay the data came from, and to which relay to send it next. The relay then rewraps the package and sends it on.

The last relay decrypts all layers of encryption and sends the original data to the intended destination without knowing the IP address of real origin source.



IP Obfuscation

Because the routing of all communication is partially hidden at every hop in the virtual circuit, the onion routing mechanism eliminates the possibility that the final communicating peers can be determined by anyone who may apply surveillance to the network.



Tor's use is intended to protect the personal privacy of users, as well as their freedom and ability to conduct confidential communication by keeping their Internet activities from being monitored.

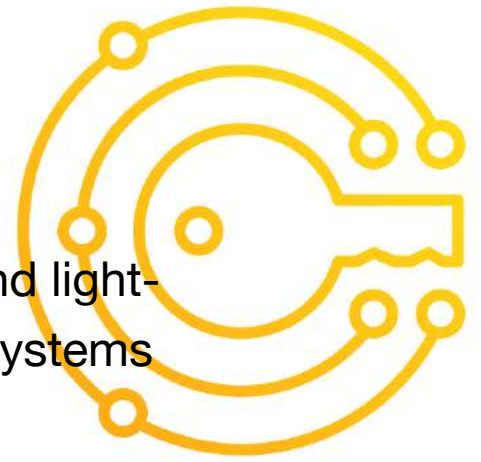
IP obfuscation functionality will be expanded by I2p technology integration within wallets. I2p (invisible internet project) is a highly obfuscated tunneling service using ipv6 that is similar to Tor, but has some other major advantages.

Instead of circuit based routing with Tor, I2P performs packet based routing that is similar to the internet's IP routing. In addition, I2p does not rely on a centralized directory service to get route information as Tor does. It uses distributed hash tables (DHTs) to coordinate the state of the network, so network routes are updated dynamically. Also, I2p establishes two independent simplex tunnels for traffic to and from each host as opposed to Tor's single duplex circuit.



Wallets

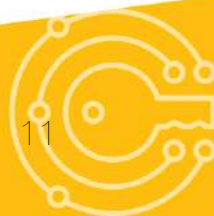
CrypticCoin offers desktop users full QT-based wallets and lightweight Electrum-based wallets for all popular operating systems (Linux, MacOS, Windows).



CrypticCoin will offer mobile users “easy-to-use” lightweight Electrum-based wallets with unique design for popular mobile operating systems (Android, iOS).

All CrypticCoin wallets support anonymity features by default through hiding the user's real IP address when making transactions, i.e. all wallets have Tor integration out of the box. So, the wallets have no built-in ability to connect to or broadcast user data over clear internet.

To increase user security, wallets have multi-signature support, which requires more than one key to authorize a transaction. Also, wallets are able to handle P2P QR-code scan transactions with instant verification. Clients are able to also import QR-codes from paper wallets to pull balances from cold storage if desired. A lightweight Electrum-based wallet does not need to download the whole blockchain, instead it requests the necessary information from secure remote Electrum servers which handle the rest of the CrypticCoin network. Electrum servers do not store user accounts with private keys. Private keys never leave user devices and are not shared with Electrum servers. Lightweight Electrum-based wallets are fast with low resource usage, have no delays for primary synchronization and are always up-to-date.

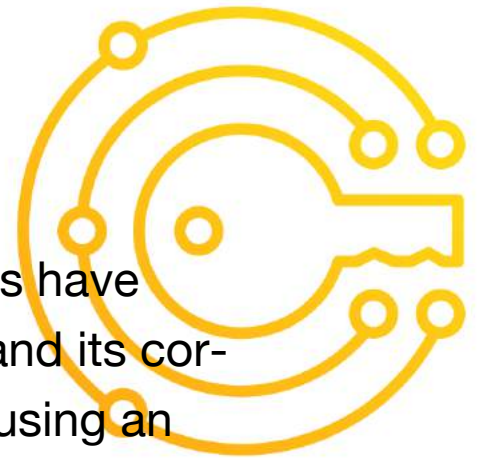


Wallets

A lightweight Electrum-based wallet helps protect users from their own mistakes and allows users to recover their wallet with a secret seed phrase. Additionally, it offers a simple and easy to use cold storage solution. This allows users to store all or part of their coins in an offline manner. While using the lightweight Electrum-based wallet, transactions are completed via Simple Payment Verification (SPV), a technique that allows for the wallet to verify transactions through proof of inclusion; a method for verifying if a particular transaction is included in a block without downloading the entire block. SPV allows for nearly instant payment confirmations because it acts as a thin client that only needs to download the block headers, which are drastically smaller than full blocks.



FreeCO



CrypticCoin is not launched through an ICO. The founders have paid for the development and launch of the CrypticCoin and its corresponding ecosystem without raising millions of dollars using an ICO. Instead of distributing the initial share of CrypticCoins only between founders, the team and advisors, CrypticCoin will reward the initial community of early adopters as well.

In the beginning, CrypticCoin will launch the Free Coin Offering (FreeCO) and give away free coins to early adopters, who will be participating in the FreeCO. Furthermore, there will be long-term post-FreeCO to incentivize community members, encourage active participants and promote CrypticCoin to make our community grow.

The amount of coins that will be distributed during FreeCO is limited to 5% of the CrypticCoin total supply. The amount of coins that will be distributed during post-FreeCO is limited to 10% of the CrypticCoin total supply.

Here at CrypticCoin it was decided to do things differently and give free coins to the community. Most importantly, we want to encourage the inclusion of newbies to the crypto space to make it more known what the potential of cryptocurrency is. It is the vision of the CrypticCoin team to build and launch something they believe in. CrypticCoin offers security, anonymity and privacy for every human being.

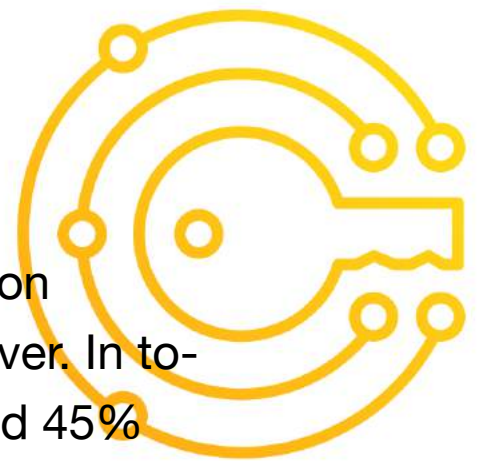


FreeCO

Free CrypticCoins will be distributed through a one-time links mechanism during FreeCO and post-FreeCO. The one-time link represents a gift certificate for some amount of free CrypticCoins (designated by a CrypticCoin admin that carries out the will of the founders and advisory team). The community members will receive their free CrypticCoins using these one-time links. All that is needed is to follow the one-time link and present the CrypticCoin address to which the corresponding amount of CrypticCoins will be transferred immediately from the FreeCo or post-FreeCo pool.



Economics



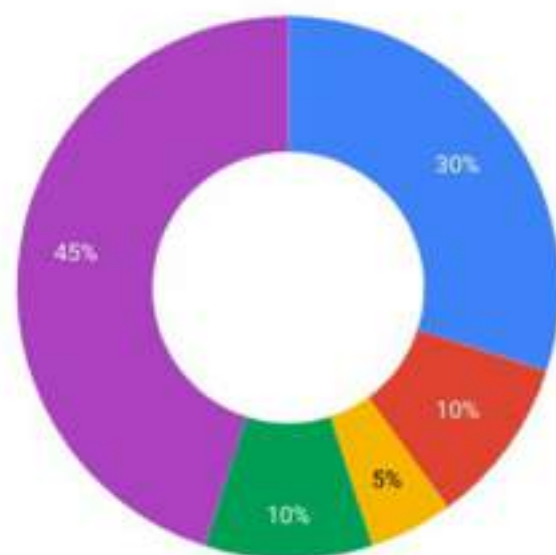
CrypticCoin launches with a capped supply. Only 7.6 billion (7,598,607,351 to be exact) CrypticCoins will be issued ever. In total CrypticCoin will have 55% pre-mined CrypticCoins and 45% CrypticCoins available for mining for about 6 years. The purposes of 55% pre-mined CrypticCoins are the following:

- 30% pre-mined CrypticCoins will be reserved for founders and team.

This amount will be transferred to 30 wallets (1% to each wallet).

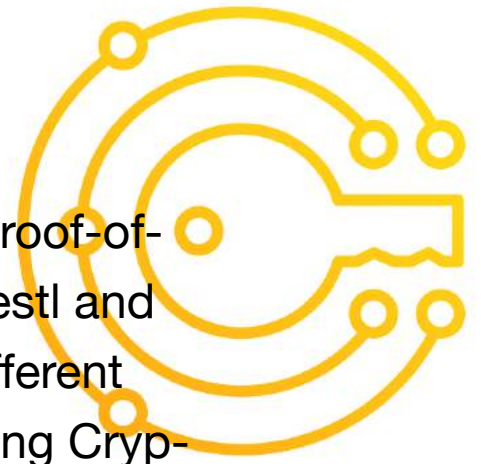
- 10% pre-mined CrypticCoins will be reserved for advisors, ambassadors and expansion and growth purposes. This amount will be transferred to 20 wallets (0.5% to each wallet).
- 5% pre-mined CrypticCoins will be reserved for FreeCO and will be transferred to a dedicated address of the FreeCO pool.
- 10% pre-mined CrypticCoins will be reserved for post-FreeCO and will be transferred to a dedicated address of the post-FreeCO pool.

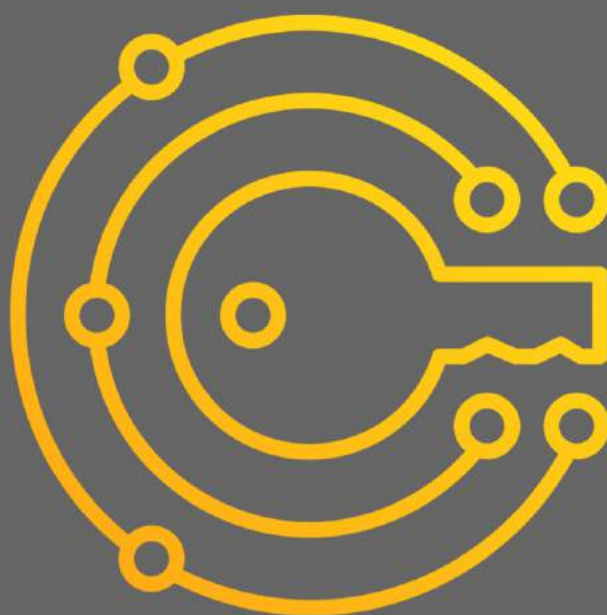
CrypticCoin Economic model



Multi-Mining

CrypticCoin supports multi-mining that combines the 5 Proof-of-Work hashing algorithms: Scrypt, Blake2s, X17, Myr-Groestl and Lyra2REv2. It means that a wide range of people with different types of mining devices have equal opportunities for mining CrypticCoins. In addition, multi-mining allows the CrypticCoin to have higher protections against Sybil Attacks compared to other cryptocurrencies, which support only single PoW hashing algorithm. All 5 mining algorithms have the same target block time and only their hash rates are impacted due to the target difficulty.





Technical Overview



WHITEPAPER
TECHNICAL OVERVIEW
<https://crypticcoin.io/>

CrypticCoin gives our users the ability to choose, compare and eventually use the confidentiality mechanism that best suits them. CrypticCoin is a coin designed for the discretion of users by offering them the appropriate crypto tools.



As a basis, we decided to implement a mix of an enriched and enhanced version of the ZeroCash Protocol (zk-SNARK systems protocol) and a Hybrid version of Verge (Stealth Addressing Technologies). However, we make no claims that CrypticCoin will offer only these two mechanisms, over time, other privacy technologies (e.g., Confidential Transactions) will be implemented in CrypticCoin.

Over time other privacy technologies will be added to CrypticCoin.

Deep view of ZeroCash Protocol

CrypticCoin set its goal to build one of the most reliable and secure privacy coins on the market. There was a decision to merge the ZeroCash protocol with others to provide multiple layers of security in a cryptocurrency. ZeroCash is an amazing cryptography protocol, thanks to the work of a group of scientists. Zerocash is a solution for private payments using cryptocurrency.

They introduced the notion of a decentralized anonymous payment scheme (*DAP scheme*), which formally captures the functionality and security guarantees of a full-fledged decentralized electronic currency with strong anonymity guarantees. The construction of this protocol has proved its high level of security under specific cryptographic assumptions. The construction leverages recent advances in the area of zero-knowledge proofs. Specifically, it uses zero-knowledge Succinct Non-interactive ARguments of Knowledge (zk-SNARKs). ZeroCash is not an extension to bitcoin protocol, but rather an independent technology with the same basic principles as blockchain and transactions.



Concept Overview : Zerocash Protocol

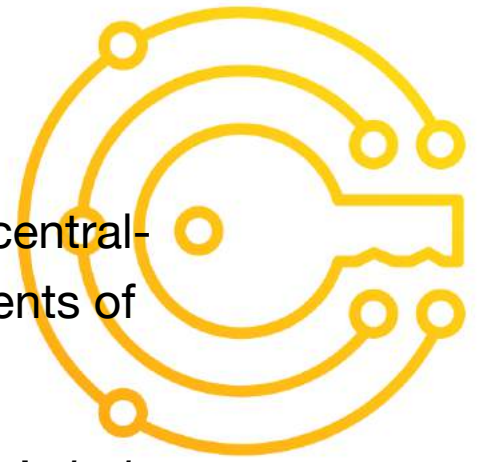


Zerocash protocol concept extends Bitcoin's protocol by adding new types of transactions that will be utilized by CrypticCoin ecosystem. The new transactions provide a separate privacy-preserving currency. This separate privacy-preserving currency's transactions reveal neither the payment's origin, destination, or amount.

Using the ZeroCash Protocol in the CrypticCoin ecosystem, CrypticCoin will create an separate anonymous currency, existing alongside a (non-anonymous) base currency, which they refer to as CrypticCoin(**CRYP**). Each user can convert (non-anonymous) CrypticCoin(**CRYP**) into (anonymous) PrivateCrypticCoin coins, which we call (**PCRYP**). Users can then send **PCRYP** to other users and split or merge **CRYP** they own in any way that preserves the total value. Users can also convert **PCRYP** back into **CRYP**, though in principle this is not necessary: all payments can be directly made in terms of **PCRYP**. Functionality is realized using just two types of transactions: Mint transactions and Pour transactions.



Decentralized anonymous payments



Decentralized anonymous payments (DAP) which is a decentralized e-cash concept that allows direct anonymous payments of any amount.

At any given time, a unique valid snapshot of the currency's *ledger* is available to all users. The ledger is a sequence of *transactions* and is *append-only*. Transactions include both the underlying currency's transactions, as well as new transactions introduced by our construction. For concreteness, they focus the discussion below on crypto currency (**though later definitions and constructions are stated abstractly**).

Step 1: User anonymity with fixed-value coins. This construction, similar to the Zerocoin protocol, shows how to obscure a payment's origin. In terms of tools, they make use of zk-SNARKs (recalled above) and a commitment scheme. Let COMM denote a statistically-hiding non-interactive commitment scheme (i.e., given randomness r and message m , the commitment is $c := \text{COMM}_r(m)$; subsequently, c is opened by revealing r and m , and one can verify that $\text{COMM}_r(m)$ equals c).

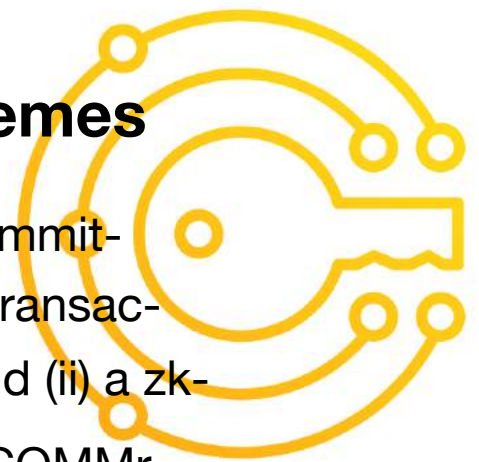
In the simplified construction, a new coin c is *minted* as follows: a user u samples a random *serial number* sn and a *trapdoor* r , computes a coin commitment $cm := \text{COMM}_r(sn)$, and sets $c := (r, sn, cm)$. A corresponding mint transaction tx_{Mint} , containing cm (but not sn or r), is sent to the ledger; tx_{Mint} is appended to the ledger only if μ has paid 1 CRYP to a backing escrow pool (e.g., the 1 CRYP may be paid via plaintext information encoded in tx_{Mint}). **Mint transactions are thus certificates of deposit, deriving their value from the backing pool.**



Decentralized anonymous payment schemes

Subsequently, letting $CMList$ denote the list of all coin commitments on the ledger, u may spend c by posting a spend transaction tx_{Spend} that contains (i) the coin's serial number sn ; and (ii) a zk-SNARK proof π of the NP statement "*I know r such that $COMMr(sn)$ appears in the list $CMList$ of coin commitments*". Assuming that sn does not already appear on the ledger (as part of a past spend transaction), μ can redeem the deposited amount of **1 CRYPT**, which μ can either keep for himself, transfer to someone else, or immediately deposit into a new coin. *(If sn does already appear on the ledger, this is considered double spending, and the transaction is discarded.)*

User anonymity is achieved because the proof π is **zero-knowledge**: while sn is revealed, no information about r is, and finding which of the numerous commitments in $CMList$ corresponds to a particular spend transaction tx_{Spend} is equivalent to inverting $f(x) := COMMr(x)$, which is assumed to be infeasible. Thus, the origin of the payment is anonymous.

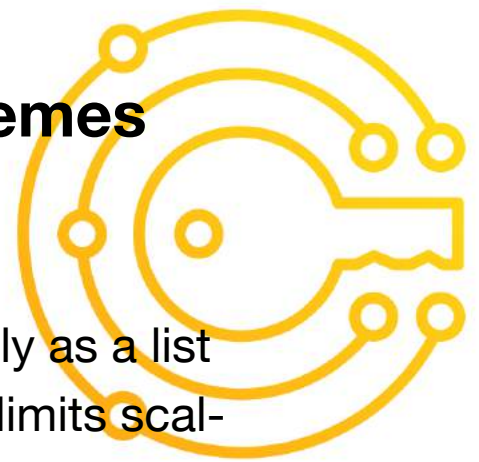


Decentralized anonymous payment schemes

Step 2: compressing the list of coin commitments.

In the previous NP statement, CMList is specified explicitly as a list of coin commitments. This naive representation severely limits scalability because the time and space complexity of most protocol algorithms (**e.g., the proof verification algorithm**) grows linearly with CMList. Moreover, coin commitments corresponding to already spent coins cannot be dropped from the CMList to reduce costs, since they cannot be identified (***due to the same zero-knowledge property that provides anonymity***).

It relies on a collision-resistant hash function CRH to avoid an explicit representation of CMList. It's maintain an efficiently updatable append-only CRH-based Merkle tree $\text{Tree}(\text{CMList})$ over the (growing) list CMList. Letting rt denote the root of $\text{Tree}(\text{CMList})$, it is well-known that updating rt to account for insertion of new leaves can be done with time and space proportional to the tree depth. Hence, the time and space complexity is reduced from linear in the size of CMList to logarithmic. With this in mind, they modify the NP statement to the following one: *"I know r such that $\text{COMM}_r(\text{sn})$ appears as a leaf in a CRH-based Merkle tree whose root is rt "*. Compared with the naive data structure for CMList, this modification increases exponentially the size of CMList which a given zk-SNARK implementation can support (concretely, using trees of depth 64, Zerocash supports 2^{64} coins).



Decentralized anonymous payment schemes

Step 3: extending coins for direct anonymous payments.

So far, the coin commitment cm of a coin c is a commitment to the coin's serial number sn . However, this creates a problem when transferring c to another user. Indeed, suppose that a user u_A created c , and u_A sends c to another user u_B . First, since u_A knows sn , the spending of c by u_B is both not anonymous (since u_A sees when c is spent, by recognizing sn) and risky (since u_A could still spend c first). Thus, u_B must immediately spend c and mint a new coin c' to protect himself. Second, if u_A in fact wants to transfer to u_B , e.g., 100BTC, then doing so is both unwieldy (since it requires 100 transfers) and not anonymous (since the amount of the transfer is leaked). And third, transfers in amounts that are not multiples of 1 BTC (the fixed value of a coin) are not supported. Thus, the simplified construction described is inadequate as a payment scheme.

They address this by modifying the derivation of a coin commitment, and using pseudorandom functions to target payments and to derive serial numbers, as follows. They use three pseudorandom functions (derived from a single one). For a seed x these are denoted $PRF_x^{addr}()$, $PRF_x^{sn}()$, and $PRF_x^{pk}()$. They assume that PRF^{sn} is moreover collision-resistant.

To provide targets for payments, they use *addresses*: each user u generates an address key pair (a_{pk}, a_{sk}) . The coins of u contain the value a_{pk} and can be spent only with knowledge of a_{sk} . A key pair (a_{pk}, a_{sk}) is sampled by selecting a random seed ask and setting $a_{pk} := PRF_{apk}^{addr}(0)$. A user can generate and use any number of address key pairs.

Decentralized anonymous payment schemes



Re-design minting to allow for greater functionality. To mint a coin c of a desired value v , the user μ first samples p , which is a secret value that determines the coin's serial number as $sn := PRF_{apk}^{sn}(p)$.

Then, μ commits to the tuple (a_{pk}, v, p) in two phases: (a) u computes $k := \text{COMMr}(a_{pk} || \rho)$ for a random r ; and then (b) u computes $cm := \text{COMMs}(v || k)$ for a random s . The minting results in a coin $c := (a_{pk}, \vartheta, \rho, r, s, cm)$ and a mint transaction $tx_{Mint} = (\vartheta, k, s, cm)$.

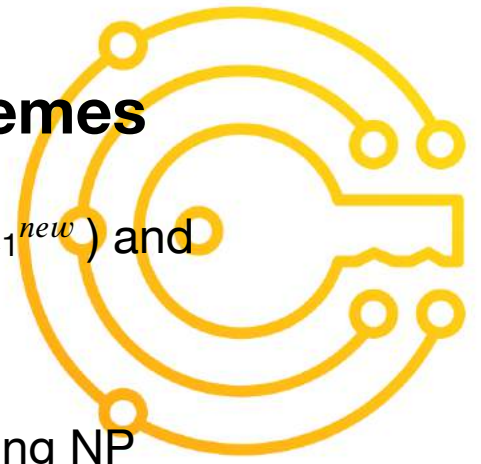
Crucially, due to the nested commitment, anyone can verify that cm in tx_{Mint} is a coin commitment of a coin of value v (by checking that $\text{COMMs}(\vartheta || k)$ equals cm) but cannot discern the owner (by learning the address key a_{pk}) or serial number (derived from p) because these are hidden in k . As before, tx_{Mint} is accepted by the ledger only if u deposits the correct amount, in this case v CRYP.

Coins are spent using the *pour* operation, which takes a set of input coins, to be consumed, and “pours” their value into a set of fresh output coins — such that the total value of output coins equals the total value of the input coins.

Suppose that μ , with address key pair $(a_{pk}^{old}, a_{sk}^{old})$, wishes to consume his coin $c_{old} = (a_{pk}^{old}, a_{sk}^{old}, p^{old}, r^{old}, s^{old}, cm^{old})$ and produce two new coins c_1^{new} and c_2^{new} , with total value $\vartheta_1^{new} + \vartheta_2^{new} = \vartheta^{old}$, respectively targeted at address public keys a_{pk1}^{new} and a_{pk2}^{new} . (The addresses a_{pk1}^{new} and a_{pk2}^{new} may belong to μ or to some other user.) The user μ , for each $i \in \{1, 2\}$, proceeds as follows: (i) μ samples serial number randomness p_i^{new} ; (ii) μ computes $k_i^{new} := \text{COMMr}^{new}(a_{pki}^{new} || p_i^{new})$ for a random r_i^{new} ; and (iii) μ computes $cm_i^{new} := \text{COMMr}^{new}(\vartheta_i^{new} || k_i^{new})$ for a random s_i^{new} .



Decentralized anonymous payment schemes



This yields the coins $c_1^{new} = (a_{pk1}^{new}, \vartheta_1^{new}, p_1^{new}, r_1^{new}, s_1^{new}, cm_1^{new})$ and $c_2^{new} = (a_{pk2}^{new}, \vartheta_2^{new}, p_2^{new}, r_2^{new}, s_2^{new}, cm_2^{new})$

Next, μ produces a zk-SNARK proof π_{POUR} for the following NP statement, which they call POUR:

“Given the Merkle-tree root rt , serial number sn^{old} , and coin commitments cm_1^{new} , cm_2^{new} , I know coins

c^{old} , c_1^{new} , c_2^{new} , and address secret key a_{sk}^{old} such that:

- *The coins are well-formed:* for c^{old} it holds that $k^{old} = \text{COMM}_r^{old}(a^{old} || p^{old})$ and $cm^{old} = \text{COMM}_s^{old}(v^{old} || k^{old})$; and similarly for c_1^{new} , and c_2^{new} .
- *The address secret key matches the public key:* $a_{pk}^{old} = \text{PRF}_{\text{apk}(old)}^{addr}(0)$.
- *The serial number is computed correctly:* $sn^{old} := \text{PRF}_{\text{apk}(old)}^{sn}(p^{old})$.
- *The coin commitment cm^{old} appears as a leaf of a Merkle-tree with root rt .*
- *The values add up:* $\vartheta_1^{new} + \vartheta_2^{new} = \vartheta^{old}$

A resulting pour transaction $\text{tx}_{\text{Pour}} := (rt, sn^{old}, cm^{new}, cm_2^{new}, \pi_{\text{POUR}})$ is appended to the ledger. (As before, the transaction is rejected if the serial number sn appears in a previous transaction.)

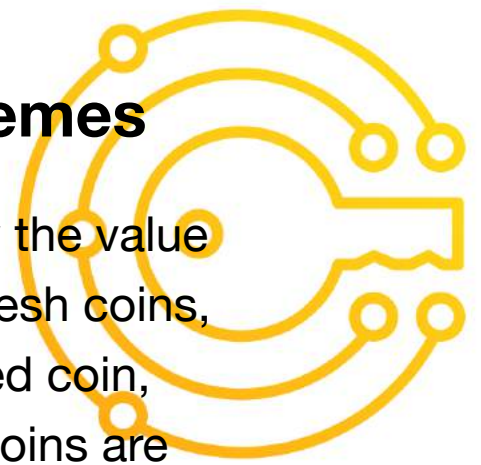
Now suppose that μ does not know, say, the address secret key a_{sk1}^{new} that is associated with the public key a_{pk1}^{new} . Then, μ cannot spend c_1^{new} , because he cannot provide a_{sk1}^{new} as part of the witness of a subsequent pour operation. Furthermore, when a user that knows a_{sk1}^{new} does spend c_1^{new} , the user μ cannot track it, because he knows no information about its revealed serial number, which is $sn_1^{new} := \text{PRF}_{\text{ask}(new)}^{sn}(p^{new})$.



Decentralized anonymous payment schemes

Also observe that tx_{Pour} reveals no information about how the value of the consumed coin was divided among the two new fresh coins, nor which coin commitment corresponds to the consumed coin, nor the address public keys to which the two new fresh coins are targeted. The payment was conducted in full anonymity.

More generally, a user may pour $N^{\text{old}} \geq 0$ coins into $N^{\text{new}} \geq 0$ coins. For simplicity they consider the case $N^{\text{old}} = N^{\text{new}} = 2$, without loss of generality. Indeed, for $N^{\text{old}} < 2$, the user can mint a coin with value 0 and then provide it as a “null” input, and for $N^{\text{new}} < 2$, the user can create (and discard) a new coin with value 0. For $N^{\text{old}} > 2$ or $N^{\text{new}} > 2$, the user can compose $\log N^{\text{old}} + \log N^{\text{new}}$ of the 2-input/2-output pours.



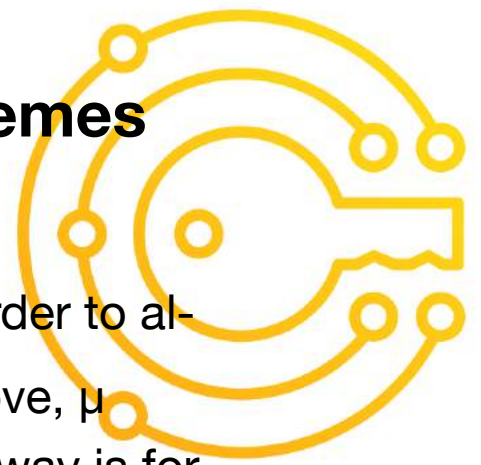
Decentralized anonymous payment schemes

Step 4: sending coins.

Suppose that a_{pk1}^{new} is the address public key of u_p . In order to allow u_1 to actually spend the new coin c^{new} produced above, u_1 must somehow send the secret values in c^{new} to u_p . One way is for u_1 to send u_p a private message, but the requisite private communication channel necessitates additional infrastructure or assumptions. They avoid this “out-of-band” channel and instead build this capability directly into construction by leveraging the ledger as follows.

Modify the structure of an address key pair. Each user now has a key pair $(addr_{pk}, addr_{sk})$, where $addr_{pk} = (a_{pk}, pk_{enc})$ and $addr_{sk} = (a_{sk}, sk_{enc})$ are generated as before. In addition, (pk_{enc}, sk_{enc}) is a key pair for a key-private encryption scheme.

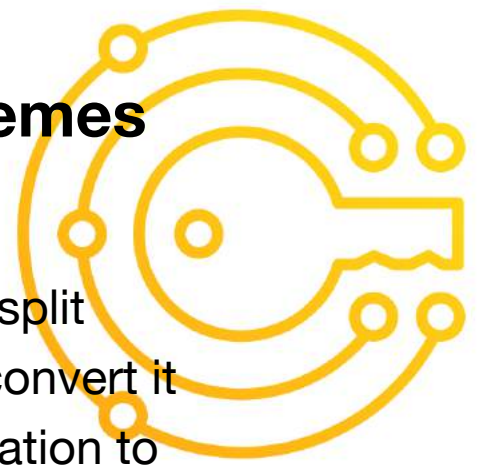
Then, u_1 computes the ciphertext C_p that is the encryption of the plaintext $(\mathfrak{g}_1^{new}, p_1^{new}, r_1^{new}, s_1^{new})$, under pk_{enc1}^{new} (which is part of u_p 's address public key $addr_{sk1}^{new}$), and includes C_p in the pour transaction tx_{Pour} . The user u_1 can then find and decrypt this message (using his sk_{enc1}^{new}) by scanning the pour transactions on the public ledger. Again, note that adding C_p to tx_{Pour} leaks neither paid amounts, nor target addresses due to the key-private property of the encryption scheme. (The user u_1 does the same with c_2^{new} and includes a corresponding ciphertext C_2 in tx_{Pour} .)



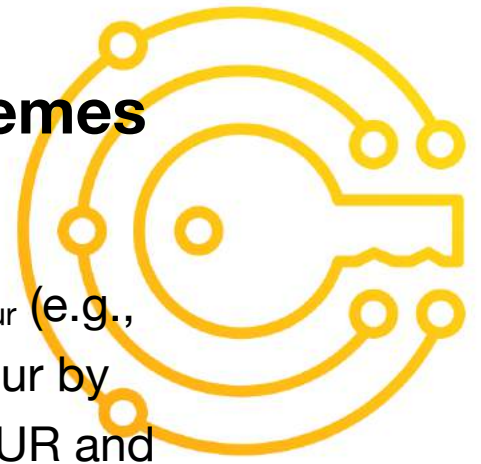
Decentralized anonymous payment schemes

Step 5: public outputs

The construction so far allows users to mint, merge, and split coins. But how can a user redeem one of his coins, i.e., convert it back to the base currency. For this, modify the pour operation to include a *public output*. When spending a coin, the user u also specifies a nonnegative v_{pub} and an arbitrary string info . The balance equation in the NP statement POUR is changed accordingly: " $\vartheta_1^{\text{new}} + \vartheta_2^{\text{new}} + \vartheta_{\text{pub}} = \vartheta^{\text{old}}$ ". Thus, of the input value ϑ^{old} , a part ϑ_{pub} is publicly declared, and its target is specified, somehow, by the string info . The string info can be used to specify the destination of these redeemed funds (e.g., a *Bitcoin wallet public key*). Both ϑ_{pub} and info are now included in the resulting pour transaction tx_{Pour} . (The public output is optional, as the user u can set $v_{\text{pub}} = 0$.)



Decentralized anonymous payment schemes



Step 6: Non-Malleability

To prevent malleability attacks on a pour transaction tx_{Pour} (e.g., embezzlement by re-targeting the public output of the pour by modifying info), they further modify the NP statement POUR and use digital signatures. Specifically, during the pour operation, the user u (i) samples a key pair $(pk_{\text{sig}}, sk_{\text{sig}})$ for a one-time signature scheme; (ii) computes $h_{\text{sig}} := \text{CRH}(pk_{\text{sig}})$; (iii) computes the two values

$h_1 := \text{PRF}_{ask1old}^{pk}(h_{\text{sig}})$ and

$h_2 := \text{PRF}_{ask2old}^{pk}(h_{\text{sig}})$, which act as MACs to “tie” h_{sig} to both address secret keys; (iv) modifies POUR to include the three values h_{sig} , h_p , h_2 and prove that the latter two are computed correctly; and (v) uses sk_{sig} to sign every value associated with the pour operation, thus obtaining a signature CT, which is included, along with pk_{sig} , in tx_{Pour} . Since the a_{ski}^{old} are secret, and with high probability h_{sig} changes for each pour transaction, the values h_1 , h_2 are unpredictable. Moreover, the signature on the NP statement (and other values) binds all of these together.

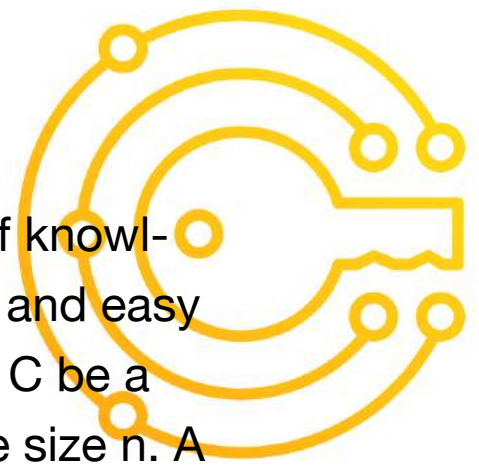


zk-SNARKs

A zk-SNARK is a non-interactive zero-knowledge proof of knowledge that is succinct, i.e., for which proofs are very short and easy to verify. More precisely, let L be an NP language, and let C be a nondeterministic decision circuit for L on a given instance size n . A zk-SNARK can be used to prove and verify membership in L , for instances of size n , as follows. After taking C as input, a trusted party conducts a one-time setup phase that results in two public keys: a proving key pk and a verification key vk . The proving key pk enables any (untrusted) prover to produce a proof π attesting to the fact that $x \in L$, for an instance x (of size n) of his choice. The non-interactive proof π is zero knowledge and a proof of knowledge. Anyone can use the verification key vk to verify the proof π ; in particular zk-SNARK proofs are publicly verifiable: anyone can verify π , without ever having to interact with the prover that generated π . Succinctness requires that (for a given security level) π has constant size and can be verified in time that is linear in $|x|$ (rather than linear in $|C|$).

The main cryptographic primitive used in this paper is a special kind of *Succinct Non-interactive ARgument of Knowledge* (SNARK). Concretely, for this using a publicly-verifiable preprocessing zero-knowledge SNARK, or zk-SNARK for short.

The main cryptographic primitive used in this paper is a special kind of *Succinct Non-interactive ARgument of Knowledge* (SNARK). Concretely, it's using a publicly-verifiable preprocessing zero-knowledge SNARK, or zk-SNARK for short.



zk-SNARKs

Informal definition

For a field F , an F -arithmetic circuit takes inputs that are elements in F , and its gates output elements in F . To model nondeterminism they consider circuits that have an input $x \in F^n$ and an auxiliary input $a \in F^h$, called a *witness*. The circuits they consider only have *bilinear gates*. Arithmetic circuit satisfiability is defined analogously to the boolean case, as follows.

Definition II.1. The *arithmetic circuit satisfiability problem* of an F -arithmetic circuit $C: F^n \times F^h \rightarrow F^i$ is captured by the relation

$R_c = \{(x, a) \in F^n \times F^h : C(x, a) = 0^i\}$; its language is $L_c = \{x \in F^n : \exists a \in F^h \text{ s.t. } C(x, a) = 0^i\}$.

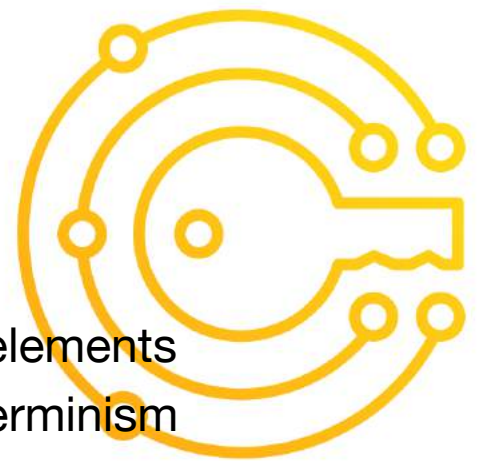
Given a field F , a (publicly-verifiable preprocessing) **zk-SNARK** for F -arithmetic circuit satisfiability is a triple of polynomial-time algorithms (KeyGen, Prove, Verify):

- $\text{KeyGen}(1^\lambda, C) \rightarrow (pk, vk)$. On input a security parameter λ (presented in unary) and an F -arithmetic circuit

C , the *key generator* KeyGen probabilistically samples a proving key pk and a *verification key* vk . Both keys are published as public parameters and can be used, any number of times, to prove/verify membership in L_c .

- $\text{Prove}(pk, x, a) \rightarrow \pi$. On input a proving key pk and any $(x, a) \in R_c$, the prover Prove outputs a non-interactive proof π for the statement $x \in L_c$.

- $\text{Verify}(vk, x, \pi) \rightarrow b$. On input a verification key vk , an input x , and a proof π , the *verifier* Verifies outputs $b = 1$ if he is convinced that $x \in L_c$.



zk-SNARKs



Completeness. For every security parameter λ , any F- arithmetic circuit C , and any $(x,a) \in R_C$, the honest prover can convince the verifier. Namely, $b = 1$ with probability $1 - \text{negl}(\lambda)$ in the following experiment: $(pk, vk) \leftarrow \text{KeyGen}(1^\lambda, C)$; $\pi \leftarrow \text{Prove}(pk, x, a)$;

$b \leftarrow \text{Verify}(vk, x, \pi)$.

Succinctness. An honestly-generated proof π has $O_\lambda(1)$ bits and $\text{Verify}(vk, x, \pi)$ runs in time $O_\lambda(|x|)$. (Here, O_λ hides a fixed polynomial factor in λ .)

Proof of knowledge (and soundness). If the verifier accepts a proof output by a bounded prover, then the prover “knows” a witness for the given instance. (In particular, soundness holds against bounded provers.) Namely, for every $\text{poly}(\lambda)$ - size adversary A , there is a $\text{poly}(\lambda)$ -size extractor E such that $\text{Verify}(vk, x, \pi) = 1$ and $(x, a) \in R_C$ with probability $1 - \text{negl}(\lambda)$ in the following experiment:

$(pk, vk) \leftarrow \text{KeyGen}(1^\lambda, C)$; $(x, \pi) \leftarrow A(pk, vk)$; $a \leftarrow E(pk, vk, \pi)$.

Perfect zero knowledge. An honestly-generated proof is perfect zero knowledge. Namely, there is a $\text{poly}(\lambda)$ -size simulator Sim such that for all stateful $\text{poly}(\lambda)$ -size distinguishers D the following two probabilities are equal:

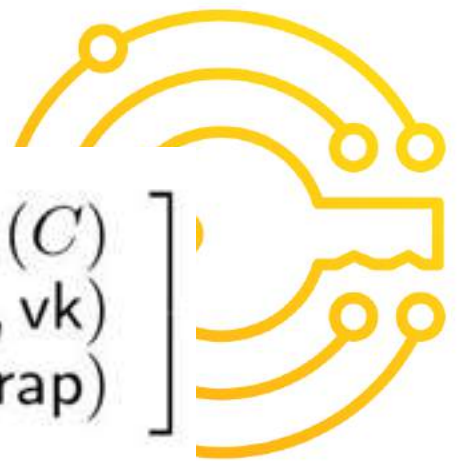
- The probability that $D(\pi) = 1$ on an honest proof.

$$\Pr \left[\begin{array}{c} (x, a) \in R_C \\ D(\pi) = 1 \end{array} \mid \begin{array}{c} (pk, vk) \leftarrow \text{KeyGen}(C) \\ (x, a) \leftarrow \mathcal{D}(pk, vk) \\ \pi \leftarrow \text{Prove}(pk, x, a) \end{array} \right]$$



- The probability that $D(n) = 1$ on a simulated proof.

$$\Pr \left[\begin{array}{l} (x, a) \in \mathcal{R}_C \\ \mathcal{D}(\pi) = 1 \end{array} \middle| \begin{array}{l} (\text{pk}, \text{vk}, \text{trap}) \leftarrow \text{Sim}(C) \\ (x, a) \leftarrow \mathcal{D}(\text{pk}, \text{vk}) \\ \pi \leftarrow \text{Sim}(\text{pk}, x, \text{trap}) \end{array} \right]$$



Known constructions and security

Security of zk-SNARKs is based on knowledge-of-exponent assumptions and variants of Diffie-Hellman assumptions in bilinear groups. While knowledge-of-exponent assumptions are fairly strong, there is evidence that such assumptions may be inherent for constructing zk-SNARKs.

zk-SNARK implementations

There are three published implementations of zk-SNARKs: (i) Parno et al. Present an implementation of zk-SNARKs for programs having no data dependencies; (ii) Ben-Sasson et al. present an implementation of zk-SNARKs for arbitrary programs (with data dependencies); and (iii) Ben-Sasson et al. present an implementation of zk-SNARKs that supports programs that modify their own code (e.g., for runtime code generation); their implementation also reduces costs for programs of larger size and allows for universal key pairs.

Each of the works above also achieves zk-SNARKs for arithmetic circuit satisfiability as a stepping stone towards their respective higher-level efforts. the implementation of provides 128 bits of security, and the field F is of a 256-bit prime order p .



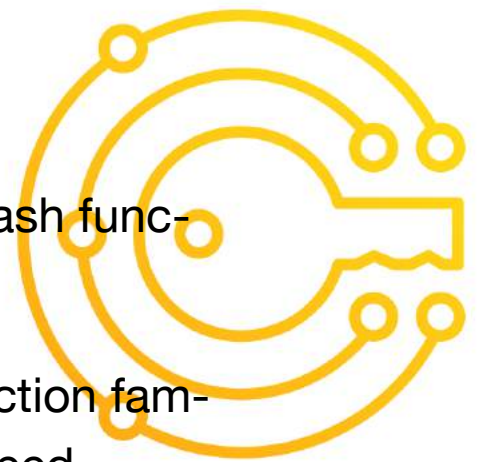
Cryptographic building blocks

Collision-resistant hashing. It's use a collision-resistant hash function $\text{CRH} : \{0,1\}^* \rightarrow \{0,1\}^{O(\lambda)}$.

Pseudorandom functions. It's use a pseudorandom function family $\text{PRF} = \{\text{PRF}_x : \{0,1\}^* \rightarrow \{0,1\}^{O(\lambda)}\}_x$ where x denotes the seed.

From PRF_x , it's derive three “non-overlapping” pseudorandom functions, chosen arbitrarily as $\text{PRF}_x^{\text{addr}}(z) := \text{PRF}_x(00||z)$, $\text{PRF}_x^{\text{sn}}(z) := \text{PRF}_x(01||z)$, $\text{PRF}_x^{\text{pk}} := \text{PRF}_x(10||z)$. Furthermore, it's assume that PRF_x^{sn} is also collision resistant, in the sense that it is infeasible to find $(x, z) \neq (x', z')$ such that $\text{PRF}_x^{\text{sn}}(z) = \text{PRF}_{x'}^{\text{sn}}(z')$.

Statistically-hiding commitments. It's use a commitment scheme COMM where the binding property holds computationally, while the hiding property holds statistically. It is denoted $\{\text{COMM}_x : \{0,1\}^* \rightarrow \{0,1\}^{O(\lambda)}\}_x$ where x denotes the commitment trapdoor. Namely, to reveal a commitment cm to a value z , it suffices to provide z and the trapdoor x ; then one can check that $\text{cm} = \text{COMM}_x(z)$.



One-time strongly-unforgeable digital signatures.

One-time strongly-unforgeable digital signatures. It's use a digital signature scheme $\text{Sig} = (\text{G}_{\text{sig}}, \text{K}_{\text{sig}}, \text{S}_{\text{sig}}, \text{V}_{\text{sig}})$ that works as follows.

- $\text{G}_{\text{sig}}(1^\lambda) \rightarrow \text{pp}_{\text{sig}}$. Given a security parameter λ (presented in unary), G_{sig} samples public parameters pp_{enc} for the encryption scheme.
- $\text{K}_{\text{sig}}(\text{pp}_{\text{sig}}) \rightarrow (\text{pk}_{\text{sig}}, \text{sk}_{\text{sig}})$. Given public parameters pp_{sig} , K_{sig} samples a public key and a secret key for a single user.
- $\text{S}_{\text{sig}}(\text{sk}_{\text{sig}}, m) \rightarrow a$. Given a secret key sk_{sig} and a message m , S_{sig} signs m to obtain a signature σ .
- $\text{V}_{\text{sig}}(\text{pk}_{\text{sig}}, m, \sigma) \rightarrow b$. Given a public key pk_{sig} , message m , and signature σ , V_{sig} outputs $b = 1$ if the signature σ is valid for message m ; else it outputs $b = 0$.

The signature scheme Sig satisfies the security property of one-time strong unforgeability against chosen-message attacks (SUF-1CMA security).

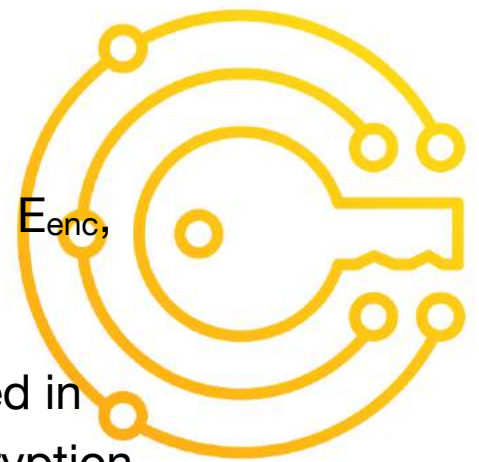


Key-private public-key encryption.

It's use a public-key encryption scheme $\text{Enc} = (\text{G}_{\text{enc}}, \text{K}_{\text{enc}}, \text{E}_{\text{enc}}, \text{D}_{\text{enc}})$ that works as follows.

- $\text{G}_{\text{enc}}(1^\lambda) \rightarrow \text{pp}_{\text{enc}}$. Given a security parameter λ (presented in unary), G_{enc} samples public parameters pp_{enc} for the encryption scheme.
- $\text{K}_{\text{enc}}(\text{pp}_{\text{enc}}) \rightarrow (\text{pk}_{\text{enc}}, \text{sk}_{\text{enc}})$. Given public parameters pp_{enc} , K_{enc} samples a public key and a secret key for a single user.
- $\text{E}_{\text{enc}}(\text{pk}_{\text{enc}}, m) \rightarrow c$. Given a public key pk_{enc} and a message m , E_{enc} encrypts m to obtain a ciphertext c .
- $\text{D}_{\text{enc}}(\text{sk}_{\text{enc}}, c) \rightarrow m$. Given a secret key sk_{enc} and a ciphertext c , D_{enc} decrypts c to produce a message m (or X if decryption fails).

The encryption scheme Enc satisfies two security properties: (i) *ciphertext indistinguishability under chosen-ciphertext attack* (IND-CCA security); and (ii) *key indistinguishability under chosen-ciphertext attack* (IK-CCA security). While the first property is standard, the second is less known; informally, IK-CCA requires that ciphertexts cannot be linked to the public key used to encrypt them, or to other ciphertexts encrypted with the same public key.



zk-SNARKs for pouring coins

As outlined in Section I-B, construction invokes a zk-SNARK for a specific NP statement, POUR. It's first recall the context motivating POUR. When a user μ pours "old" coins c_1^{old}, c_2^{old} into new coins c_1^{new}, c_2^{new} , a corresponding pour transaction

$$tx_{\text{pour}} = (rt, sn_1^{old}, sn_2^{old}, cm_1^{new}, cm_2^{new}, \mathfrak{g}_{\text{pub}}, \text{info}, *)$$

is generated. . Concretely, tx_{Pour} should demonstrate that (i) μ owns c_1^{old}, c_2^{old} ; (ii) coin commitments for c_1^{old}, c_2^{old} appear somewhere on the ledger; (iii) the revealed serial numbers sn_1^{old}, sn_2^{old} are of c_1^{old}, c_2^{old} (iv) the revealed coin commitments cm_1^{new}, cm_2^{new} are of c_1^{new}, c_2^{new}

(v) balance is preserved. Construction achieves this by including a zk-SNARK proof π_{POUR} for the statement POUR which checks the above invariants (as well as others needed for non-malleability).

The statement POUR

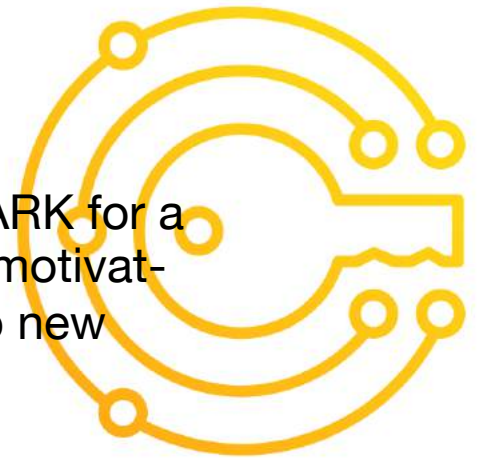
Concretely, the NP statement POUR is defined as follows.

- Instances are of the form $X = (rt, sn_1^{old}, sn_2^{old}, cm_1^{new}, cm_2^{new}, \mathfrak{g}_{\text{pub}}, h_{\text{sig}}, h_1, h_2)$. Thus, an instance x specifies a root rt for a CRH-based Merkle tree (over the list of commitments so far), the two serial numbers of the consumed coins, two coin commitments for the two new coins, a public value, and fields h_{sig}, h_1, h_2 used for non-malleability.

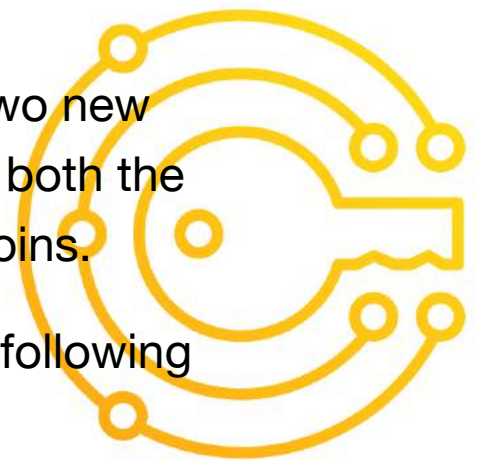
Witnesses are of the form $a = (\text{path}_1, \text{path}_2, c_1^{old}, c_2^{old}, \text{addr}_{sk1}^{old}, \text{addr}_{sk2}^{old}, c_1^{new}, c_2^{new})$ where, for each $l \in \{1, 2\}$:

$$\begin{aligned} c_i^{old} &= (\text{addr}_{pk,i}^{old}, v_i^{old}, \rho_i^{old}, r_i^{old}, s_i^{old}, cm_i^{old}) , \\ c_i^{new} &= (\text{addr}_{pk,i}^{new}, v_i^{new}, \rho_i^{new}, r_i^{new}, s_i^{new}, cm_i^{new}) \\ &\quad \text{for the same } cm_i^{new} \text{ as in } \vec{x}, \end{aligned}$$

$$\begin{aligned} \text{addr}_{pk,i}^{old} &= (a_{pk,i}^{old}, pk_{enc,i}^{old}) , \\ \text{addr}_{pk,i}^{new} &= (a_{pk,i}^{new}, pk_{enc,i}^{new}) , \\ \text{addr}_{sk,i}^{old} &= (a_{sk,i}^{old}, sk_{enc,i}^{old}) . \end{aligned}$$



Thus, a witness specifies authentication paths for the two new coin commitments, the entirety of coin information about both the old and new coins, and address secret keys for the old coins.



Given a POUR instance X , a witness a is valid for X if the following holds:

- 1) For each $i \in \{1, 2\}$:
 - a) The coin commitment cm_i^{new} of c_i^{old} appears on the ledger, i.e., $path_i$ is a valid authentication path for leaf cm_i^{new} with respect to root rt , in a CRH-based Merkle tree.
 - b) The address secret key a_{ski}^{old} matches the address public key of c_i^{old} , i.e., $apd = PRF_{aski}^{addrold}(0)$.
 - c) The serial number sn_i^{old} of c_i^{old} is computed correctly, i.e., $sn_i^{old} = PRF_{askiold}^{snold}(p_i^{old})$.
 - d) The coin c_i^{old} is well-formed, i.e., $cm_i^{old} \in \text{COMMS}^{old}(a_{pk,i}^{old} || p_i^{old} || \vartheta_i^{old})$.
 - e) The coin c_i^{new} is well-formed, i.e., $cm_i^{new} \in \text{COMMS}^{new}(a_{pk,i}^{new} || p_i^{new} || \vartheta_i^{new})$.
 - f) The address secret key $a_{pk,i}^{old}$ ties h_{sig} to $hi = PRF_{aski}^{skold}(h_{sig})$.
 - g) Balance is reserved: $\vartheta_1^{new} + \vartheta_2^{new} + \vartheta_{pub} = \vartheta_1^{old} + \vartheta_2^{old}$ (with $\vartheta_1^{old} + \vartheta_2^{old} \geq 0$ and $\vartheta_1^{old} + \vartheta_2^{old} \leq \vartheta_{max}$).

Recall that in this paper zk-SNARKs are relative to the language of arithmetic circuit satisfiability thus, in POUR via an arithmetic circuit, denoted C_{POUR} . In particular, the depth d_{tree} of the Merkle tree needs to be hardcoded in C_{POUR} , and it a parameter of construction (see below); the maximum number of supported coins is then $2^{d_{tree}}$.



ZeroCash

ZeroCash is a concrete implementation, at 128 bits of security, of DAP scheme construction. ZeroCash entails carefully instantiating the cryptographic ingredients of the construction to ensure that the zk-SNARK, the “heaviest” component, is efficient enough in practice. In the construction, the zk-SNARK is used to prove/verify a specific NP statement: POUR. While zk-SNARKs are asymptotically efficient, their concrete efficiency depends on the arithmetic circuit C that is used to decide the NP statement. Thus, a search of instantiations was made for which can be designed a relatively-small arithmetic circuit C_{POUR} for verifying the NP statement POUR.

Instantiation of building blocks

CRH, PRF, COMM from SHA256. Let H be the SHA256 compression function, which maps a 512-bit input to a 256-bit output. Rely on H , rather than the “full” hash, since this suffices for fixed-size single-block inputs, and it simplifies the construction of C_{POUR} . Instantiate CRH, PRF, COMM via H (under suitable assumptions on H).

First, instantiate the collision-resistant function CRH as $H(z)$ for $z \in \{0,1\}^{512}$; this function compresses “two-to-one”, so it can be used to construct binary Merkle trees.

Next, instantiate the pseudorandom function $\text{PRF}_x(z)$ as $H(x||z)$, with $x \in \{0,1\}^{256}$ as the seed, and $z \in \{0,1\}^{256}$ as the input. Thus, the derived functions are:

$\text{PRF}_x^{\text{addr}}(z) := H(x||00||z)$, $\text{PRF}_x^{\text{sn}}(z) := H(x||01||z)$, $\text{PRF}_x^{\text{pk}}(z) := H(x||10||z)$,

with $x \in \{0,1\}^{256}$ and $z \in \{0,1\}^{254}$.



As for the commitment scheme COMM, it only used in the following pattern:

$$k := \text{COMM}_r(a_{pk} \parallel p) ,$$
$$cm := \text{COMM}_s(v \parallel k) .$$

Due to instantiation of PRF, a_{pk} is 256 bits. So p also can be set to 256 bits and r to $256 + 128 = 384$ bits; then it is possible to compute

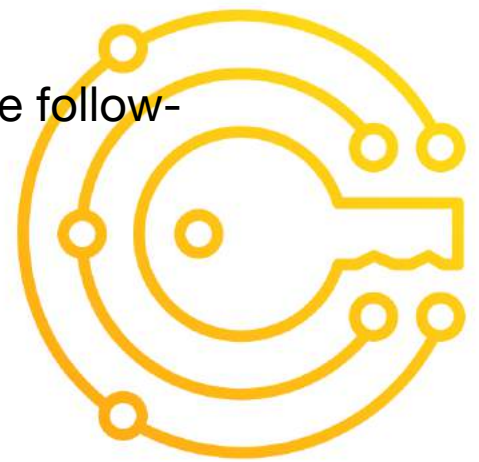
$$k := \text{COMM}_r(a_{pk} \parallel p) \text{ as } H(r \parallel [H(a_{pk} \parallel p)]_{128}) .$$

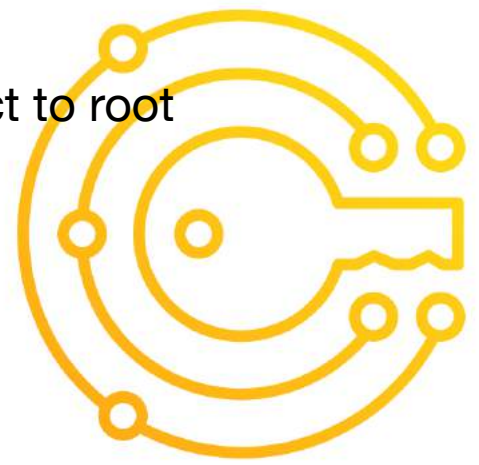
Above, $[\cdot]_{128}$ denotes the truncating of the 256-bit string to 128 bits (by dropping least- significant bits, as in the implementation). Heuristically, for any string $z \in \{0,1\}^{128}$, the distribution induced by $H(r \parallel z)$ is 2^{-128} close to uniform, and this forms the basis of the statistically-hiding property. For computing cm , set coin values to be 64-bit integers (so that, in particular, $v_{\max} = 2^{64} - 1$ in the implementation), and then compute

$$cm := \text{COMM}_s(v \parallel k) \text{ as } H(k \parallel 0^{192} \parallel v) .$$

Noticeably, the above commitment randomness s is ignored. The reason is that already known that k , being the output of a statistically-hiding commitment, can serve as randomness for the next commitment scheme.

Instantiating the NP statement POUR. The above choices imply a concrete instantiation of the NP statement POUR. Specifically, in the implementation, POUR checks that the following holds, for each $i \in \{1, 2\}$:





- path_i is an authentication path for leaf cm_i^{old} with respect to root rt , in a CRH-based Merkle tree;
- $a_{\text{pk},i}^{\text{old}} = H(a_{\text{sk},i}^{\text{old}} \| 0^{256})$;
- $\text{sn}_i^{\text{old}} = H(a_{\text{sk},i}^{\text{old}} \| 01 \| [\text{p}_i^{\text{old}}]_{254})$;
- $\text{cm}_i^{\text{old}} = H(H(r_i^{\text{old}} \| [H(a_{\text{pk},i}^{\text{old}} \| \text{p}_i^{\text{old}})]_{128}) \| 0^{192} \| v_i^{\text{old}})$;
- $\text{cm}_i^{\text{new}} = H(H(r_i^{\text{new}} \| [H(a_{\text{pk},i}^{\text{new}} \| \text{p}_i^{\text{new}})]_{128}) \| 0^{192} \| v_i^{\text{new}})$; and
- $h_i = H(a_{\text{sk},i}^{\text{old}} \| 10 \| b_i \| [h_{\text{Sig}}]_{253})$ where $b_1 := 0$ and $b_2 := 1$.

Moreover, POUR checks that $v_1^{\text{new}} + v_2^{\text{new}} + v_{\text{pub}} = v_1^{\text{old}} + v_2^{\text{old}}$, with $v_1^{\text{old}}, v_2^{\text{old}} > 0$ and $v_1^{\text{old}} + v_2^{\text{old}} < 2^{64}$.

Finally, as mentioned, in order for C_{POUR} to be well-defined, need to fix a Merkle-tree depth d_{tree} . In the implementation, fix $d_{\text{tree}} = 64$, and thus support up to 2^{64} coins.

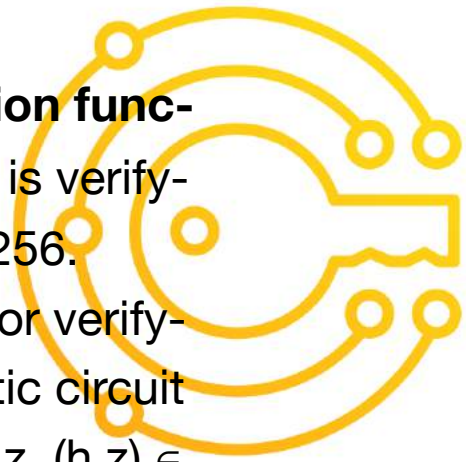
Instantiating Sig. For the signature scheme Sig, the ECDSA is used. However, standard ECDSA is malleable: both (r, s) and $(r, -s)$ verify as valid signatures. A non-malleable variant is used, where s is restricted to the “lower half” of field elements.

Instantiating Enc. For the encryption scheme Enc, the key-private Elliptic-Curve Integrated Encryption Scheme (ECIES) is used; it is one of the few standardized key-private encryption schemes with available implementations.

Arithmetic circuit for pouring coins

DAP scheme construction also requires zk-SNARKs relative to the NP statement POUR. These are obtained by invoking a zk-SNARK for arithmetic circuit satisfiability on an arithmetic circuit C_{POUR} , which verifies the NP statement POUR.





An arithmetic circuit for verifying SHA256's compression function. The vast majority of the “verification work” in POUR is verifying computations of H , the compression function of SHA256. Thus, begin with construction of an arithmetic circuit C_H for verifying SHA256 computations. Need to construct an arithmetic circuit C_H such that, for every 256-bit digest h and 512-bit input z , $(h, z) \in R_{C_H}$ if and only if $h = H(z)$. Naturally, goal is to minimize the size of C_H . The strategy is to construct C_H , piece by piece, by closely following the SHA256 official specification. For each subcomputation of SHA256, nondeterminism and field operations are used to verify the subcomputation using as few gates as possible.

Overview of SHA256's compression function. The primitive unit in SHA256 is a 32-bit *word*. All subcomputations are simple word operations: three bitwise operations (and, or, xor), shift-right, rotate-right, and addition modulo 2^{32} . The compression function internally has a state of 8 words, initialized to a fixed value, and then transformed in 64 successive rounds by following the 64-word *message schedule* (deduced from the input z). The 256-bit output is the concatenation of the 8 words of the final state.

Representing a state. For each word operation (except for addition modulo 2^{32}), it is more efficient to verify the operation when its inputs are represented as separate wires, each carrying a bit. Thus, C_H maintains the 8-word state as 256 individual wires, and the 64-word message schedule as $64 \cdot 32$ wires.

Addition modulo 32. To verify addition modulo 2^{32} the following technique is used. Given two words A and B , need to compute:

$$\alpha := \sum_{i=0}^{31} 2^i (A_i + B_i).$$



Because F has characteristic larger than 2^{33} , there is no wrap around; thus, field addition coincides with integer addition. Then make a non-deterministic guess for the 33 bits a_i of a (including carry), and enforce consistency by requiring that:

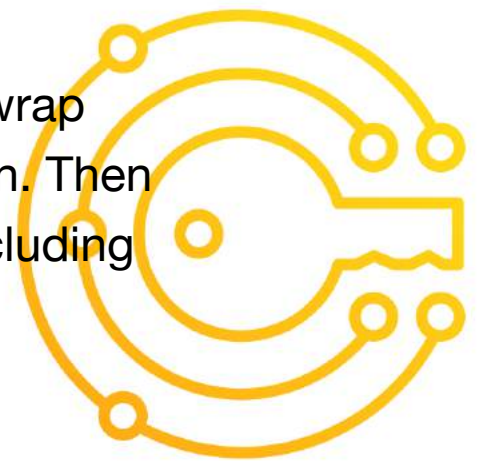
$$\alpha = \sum_{i=0}^{32} 2^i \alpha_i$$

To ensure that each $a_i \in \{0,1\}$, use a 33-gate subcircuit computing $a_i(a_i - 1)$, all of which must be 0 for the subcircuit to be satisfiable. Overall, verifying addition modulo 2^{32} only requires 34 gates. This approach extends in a straightforward way to summation of more than two terms.

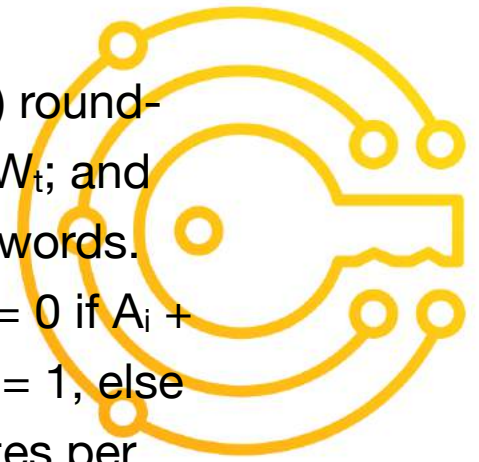
Verifying the SHA256 message schedule. The first 16 words W_i of the message schedule are the 16 words of the 512-bit input z . The remaining 48 words are computed as $W_t := a_1(W_{t-2}) + W_{t-7} + a_0(W_{t-15}) + W_{t-16}$, where $a_0(W) := \text{rotr}_7(W) \oplus \text{rotr}_{18}(W) \oplus \text{shr}_3(W)$ and a_1 has the same structure but different rotation and shift constants.

The rotation and shift amounts are constants, so rotates and shifts can be achieved by suitable wiring to previously computed bits (or the constant 0 for high-order bits in shr). Thus, since the XOR of 3 bits can be computed using 2 gates, both a_0 and a_1 can be computed in 64 gates. Then compute (or more precisely, guess and verify) the addition modulo 2^{32} of the four terms.

Verifying the SHA256 round function. The round function modifies the 8-word state by changing two of its words and then permuting the 8-word result.



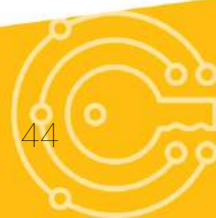
Each of the two modified words is a sum modulo 2^{32} of (i) round-specific constant words K_t ; (ii) message schedule words W_t ; and (iii) words obtained by applying simple functions to state words. Two of those functions are bitwise majority ($\text{Maj}(A, B, C)_i = 0$ if $A_i + B_i + C_i < 1$ else 1) and bitwise choice ($\text{Ch}(A, B, C)_i = B_i$ if $A_i = 1$, else C_i). Need to verify correct computation of Maj using 2 gates per output bit, and Ch with 1.

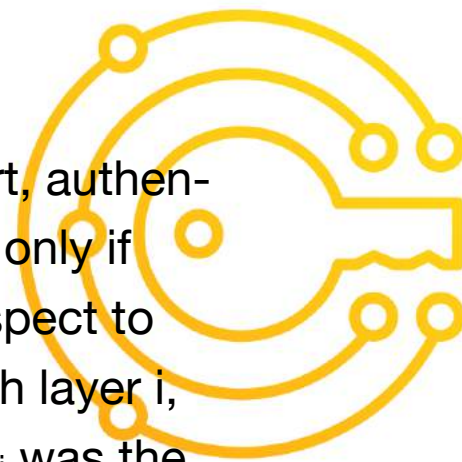


Then, instead of copying 6 unchanged state words to obtain the permuted result, make the permutation implicit in the circuit's wiring, by using output wires of previous sub-computations (sometimes reaching 4 round functions back) as input wires to the current sub-computation.

Arithmetic circuit for POUR

The NP statement POUR requires verifying membership in a Merkle tree based on H , a few additional invocations of H , and integer addition and comparison. Need to construct the circuit C_{POUR} for POUR by combining various subcircuits verifying each of these. There remains to discuss the subcircuits for verifying membership in a Merkle tree (using the aforementioned subcircuit C_H for verifying invocations of H), and integer addition and comparison.





Merkle tree membership.

Need to construct an arithmetic circuit that, given a root rt , authentication *path* and coin commitment cm , is satisfied if and only if *path* is a valid authentication path for the leaf cm with respect to the root rt . The authentication path *path* includes, for each layer i , an auxiliary hash value h_i and a bit r_i specifying whether h_i was the left ($r_i = 0$) or the right ($r_i = 1$) child of the parent node. Then check membership in the Merkle tree by verifying invocations of H , bottom-up. Namely, for $d = 64$, set $k_{d-1} = cm$; then, for each $i = d-1, \dots, 1$, set $B_i = h_i \parallel k_i$ if $r_i = 0$ else $k_i \parallel h_i$, and compute $k_{i-1} = H(B_i)$. Finally check that the root k_0 matches the given root rt .

Integer addition.

Need to construct an arithmetic circuit that, given 64-bit integers A, B, C (presented as binary strings), is satisfied if and only if $C = A + B$ over the integers. Again relying on the fact that F 's characteristic is sufficiently large, do so by checking that:

$$\sum_{i=0}^{63} 2^i c_i = \sum_{i=0}^{63} 2^i (b_i + a_i) \text{ over } \mathbb{F};$$

this is enough, because there is no wrap around.

Integer comparison

Need to construct an arithmetic circuit that, given two 64-bit integers A, B (represented in binary), is satisfied if and only if $A + B$ fits in 64 bits (i.e. $A + B < 2^{64}$). Do so by checking that

$$\sum_{i=0}^{63} 2^i (b_i + a_i) = \sum_{i=0}^{63} c_i$$

for some $c_i \in \{0, 1\}$. Indeed, if $A + B < 2^{64}$ then it suffices to take c_i as the binary representation of $A + B$.



However, if $A + B > 2^{64}$ then no choice of c_i can satisfy the constraint as:

$$\sum_{i=0}^{63} c_i \leq 2^{64} - 1.$$

Overall, this requires 65 gates (1 gate for the equality check, and 64 gates for ensuring that c_0, \dots, c_{63} are boolean).

Overall circuit sizes.

See Figure for the size of C_{POUR} . More than 99% of the gates are devoted to verifying invocations of H .

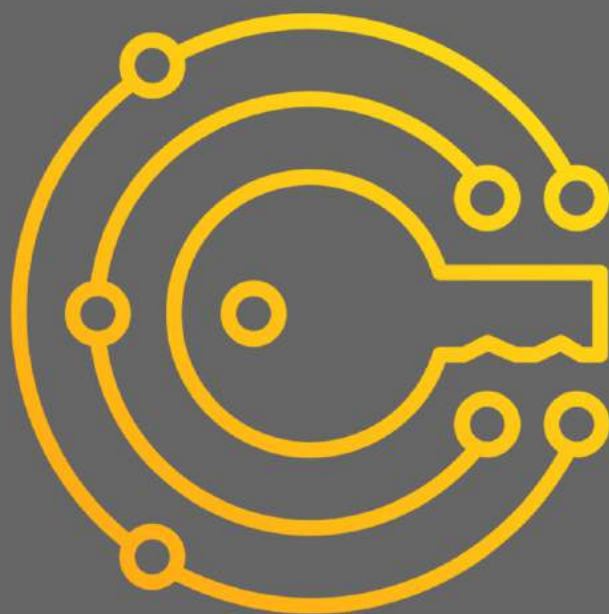
Gate count for C_{POUR}	
Ensure cm_1^{old} is in Merkle tree (1 layer out of 64)	1 802 304 (28 161)
Ensure cm_2^{old} is in Merkle tree (1 layer out of 64)	1 802 304 (28 161)
Check computation of $sn_1^{\text{old}}, sn_2^{\text{old}}$	$2 \times 27\,904$
Check computation of $a_{pk,1}^{\text{old}}, a_{pk,2}^{\text{old}}$	$2 \times 27\,904$
Check computation of $cm_1^{\text{old}}, cm_2^{\text{old}}, cm_1^{\text{new}}, cm_2^{\text{new}}$	$4 \times 83\,712$
Check computation of h_1, h_2	$2 \times 27\,904$
Ensure that $v_1^{\text{new}} + v_2^{\text{new}} + v_{\text{pub}} = v_1^{\text{old}} + v_2^{\text{old}}$	1
Ensure that $v_1^{\text{old}} + v_2^{\text{old}} < 2^{64}$	65
Miscellaneous	2384
Total	4 109 330

Figure: Size of the circuit C_{POUR} , which verifies the statement POUR.construct





Figure: Construction of a DAP scheme using zk-SNARKs and other ingredients



Technology Roadmap



WHITEPAPER
TECHNICAL OVERVIEW
<https://crypticcoin.io/>

Roadmap (Future Plans)



CrypticCoin will be constantly improved by implementing new features and expanding its ecosystem. Such continuous development is performed by the core development team that consists of several permanent contributors. The development of the following tasks are scheduled:

- Forum platform that allows users to suggest any ideas that will be useful for the CrypticCoin community. Such ideas will be assessed by the community through a voting mechanism within the CrypticCoin forum. The ideas selected by the voting will be crowdfunded and implemented.
- Official mining pool that will support mining based on any of 5 Proof-of-Work hashing algorithms used in CrypticCoin (Scrypt, Blake2s, X17, Myr-Groestl and Lyra2REv2).
- Unique and easy to use mobile wallets for Android and iOS platforms. For CrypticCoin participants, these wallets will be as user-friendly as possible.
- Wallets with built-in I2p integration. The wallets have improved anonymity features and will be offered to CrypticCoin users for more robust IP obfuscation.
- Encrypted p2p chat between CrypticCoin network members. Instant messaging system that ensures encryption and privacy of P2P (Peer-to-Peer) communications.
- RSK smart contracts integration. RSK (Rootstock) is a two-way pegged sidechain that extends CrypticCoin by adding the smart contracts functionality.
- Launching the company for the debit cards connecting the virtual card and plastic card. Issuing the debit cards, which will support fiat currencies (EUR, USD, etc.), the CrypticCoin, other cryptocurrencies and will have native exchange capability between supported currencies. CrypticCoin development roadmap is presented below.

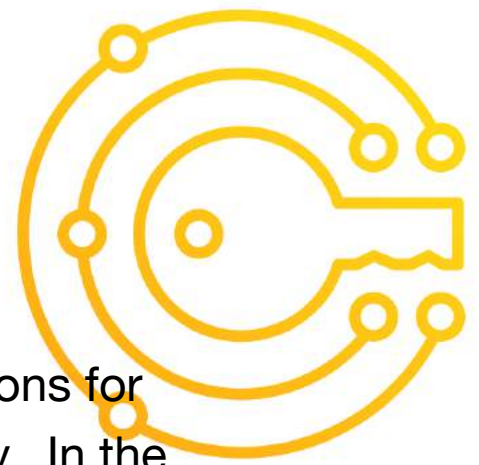
The roadmap can be slightly changed by adding additional tasks and rescheduling current and new tasks.



Roadmap (Future Plans)

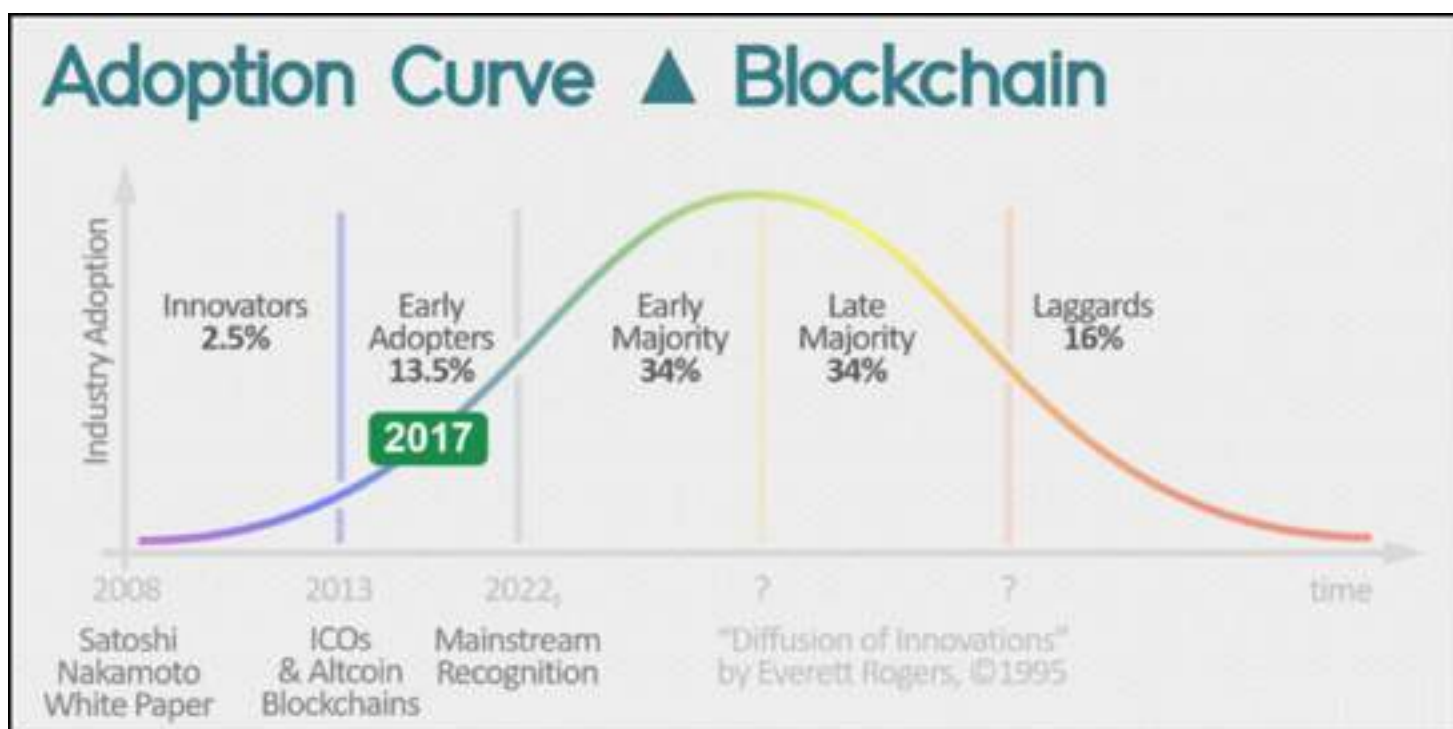


Roadmap (Future Plans)



Website Transaction Inclusion

Our intention is for early adopters to have increasing options for spending and also expanding the CrypticCoin community. In the future we would like more websites to accept CRYP as another form of payment. Currently a few sites (Etsy/Overstock) chose to accept only certain cryptocurrencies, but this is also an opportunity for vendors to add a revenue stream.

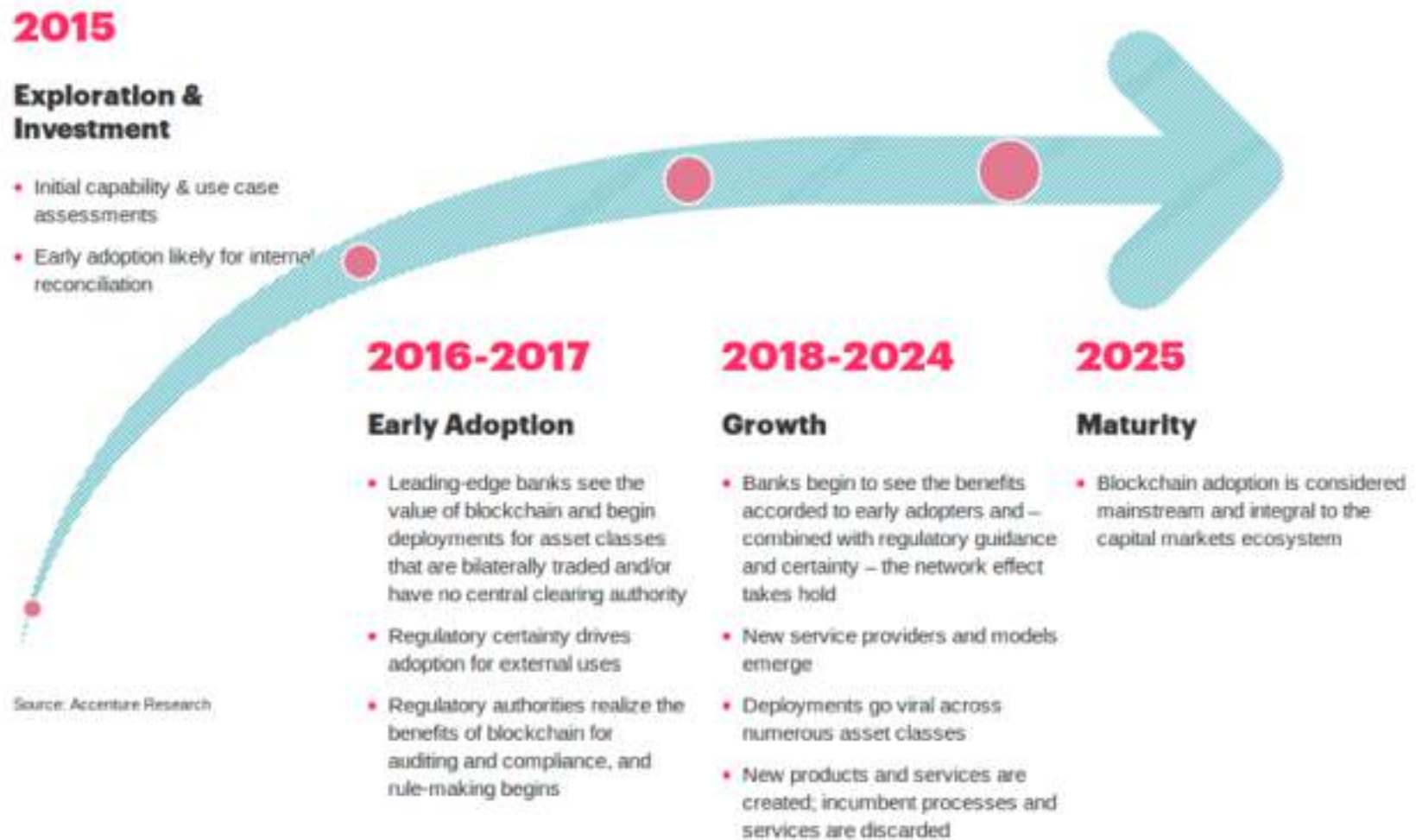
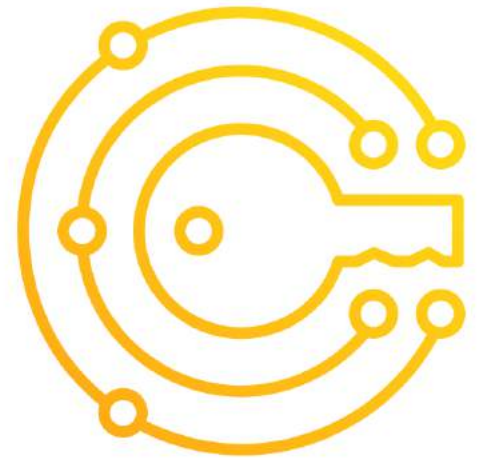


Marketplace creation:

Push for acceptance of CRYP online and offline. CrypticCoin has plans for working with vendors to utilize the token as a means of exchange.



Roadmap (Future Plans)



This diagram shows the expansion of the market from exploration and investment all the way to maturity. Most are not aware that we are in an early adoption mode.



Roadmap (Future Plans)

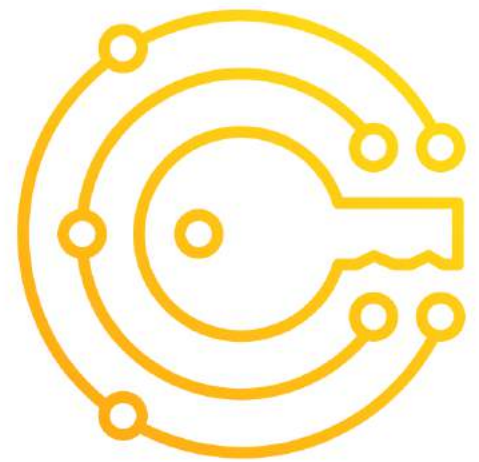
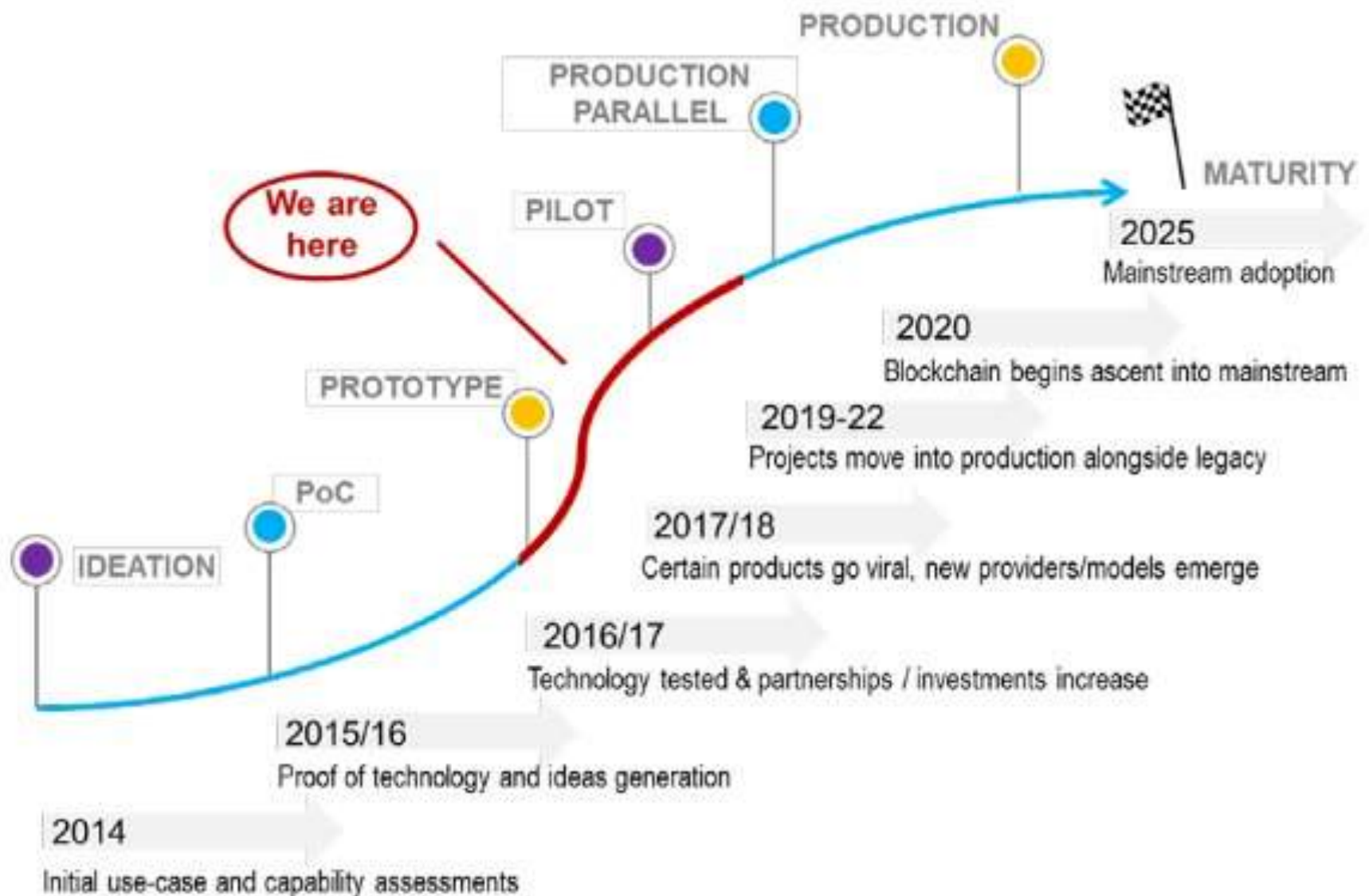


Figure 45: Development timeline – where are we now?

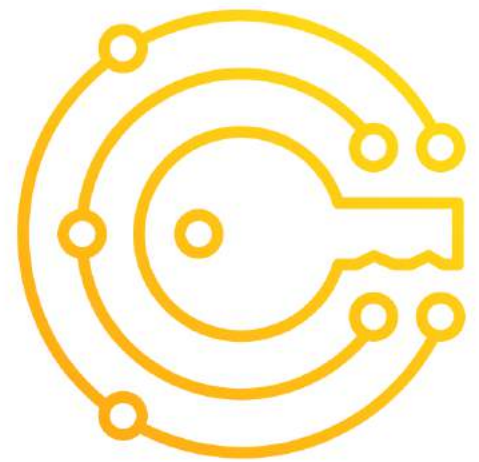


Source: Accenture, Credit Suisse estimates

Featured here is another diagram showing where we are currently in the Blockchain space. CrypticCoin is in position for the best time for growth and expansion of blockchain technology. **This timeline featured here is simply the Development timeline of BLOCKCHAIN TECHNOLOGY IN GENERAL, and not specifically CrypticCoin.**



Privacy Coin Comparison



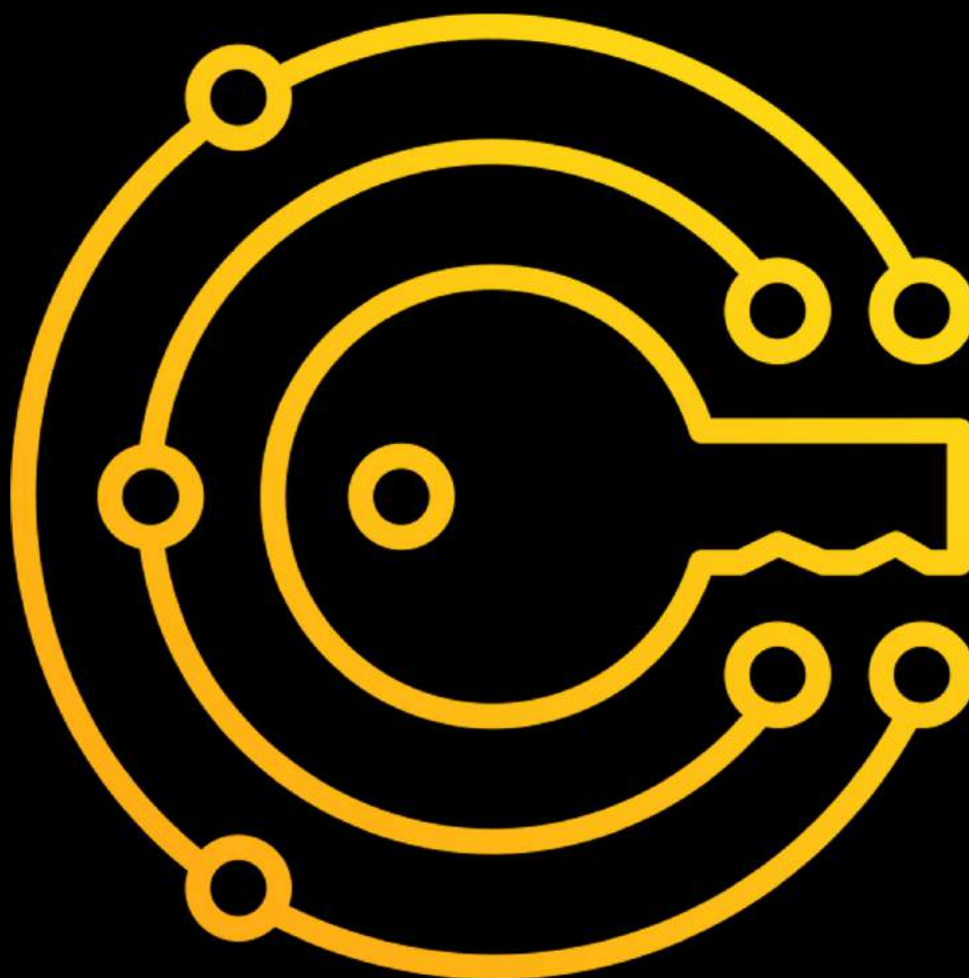
8 PRIVACY COINS : 2018

A DETAILED BREAKDOWN OF 7 PRIVACY COINS COMPARED TO THE CRYPTICCOIN FRAMEWORK

PRIVACY COIN DETAILS	CrypticCoin	Bitcoin	Monero	ZCash	ZCoin	PIVX	Navcoin	Verge
Circ. Supply - Millions	7,600	16.8	15.8	3.5	4.3	55.7	62.5	14,700
Total Supply - Millions	7,600	21	16	3.5	4.2	55.7	62.5	14,500
Max Supply - Millions	7,600	21	⊗	21	21	⊗	⊗	16,500
Block Time - Seconds	30	600	120	150	600	60	30	30
PoW Mining	✓	✓	✓	✓	✓	⊗	⊗	✓
PoS Staking	⊗	⊗	⊗	⊗	⊗	✓	✓	⊗
I2P	✓	⊗	⊗	⊗	⊗	✓	✓	✓
Masternodes	⊗	⊗	⊗	⊗	✓	✓	⊗	⊗
Privacy Tech	Mixing	⊗	Ring CT	ZK-Snarks	ZeroCoin	ZeroCoin	Dual Block-TOR	Mixing
Native TOR	✓	⊗	⊗	⊗	⊗	⊗	⊗	✓
OBFS4	✓	⊗	⊗	⊗	⊗	⊗	⊗	✓
ZeroCash Protocol	✓	⊗	⊗	✓	⊗	⊗	⊗	⊗
Stealthed IP	✓	⊗	⊗	⊗	⊗	⊗	✓	✓
Stealth Addresses	✓	⊗	✓	✓	✓	✓	⊗	✓
Stealth Send	✓	⊗	✓	✓	⊗	⊗	✓	✓
Nodes Online / 24 hrs	?	12100	2,900	1,200	1,700	2,100	280	?
Voting Governance	⊗	⊗	⊗	⊗	⊗	✓	✓	⊗
Mobile Wallets	Q2	✓	⊗	⊗	✓	✓	✓	✓
Light Wallet	✓	✓	✓	✓	⊗	✓	✓	✓
Hardware Wallet	⊗	✓	⊗	✓	⊗	⊗	✓	⊗
Avg Transaction Fee	0.01 CRTP	0.0003 BTC	0.01 XMR	0.001 ZEC	0.13 XZC	0.003 PIVX	0.0003 NAV	0.01 XVG
Development Fund	✓	✓	⊗	✓	✓	✓	✓	⊗
Smart Contracts	Q4	RSC beta	⊗	⊗	⊗	⊗	⊗	?
CRYPTICCOIN.IO								
No ICO	✓	✓	✓	✓	✓	✓	✓	✓
Market Cap	Unknown	\$152,178,780,330	\$3,162,716,223	\$782,014,139	\$151,413,609	\$205,790,470	\$65,261,594	\$430,892,442
Price	Unknown	\$8.888	\$199.74	\$223.44	\$34.99	\$3.69	\$1.04	\$0.029271
All Time High	Unknown	\$20,000	\$494	\$926	\$153	\$14	\$4.71	\$0.27
Possible x Gain	Unknown	2.25	2.4	4.1	4.3	3.7	4.5	9.2

The privacy coin comparison above articulates a brief comparison between CrypticCoin and many other existing privacy coin options in the market. It is our belief that CrypticCoin will grow and evolve to stand toe-to-toe and even surpass the coins featured in this list.





WHITEPAPER
TECHNICAL OVERVIEW
<https://crypticcoin.io/>