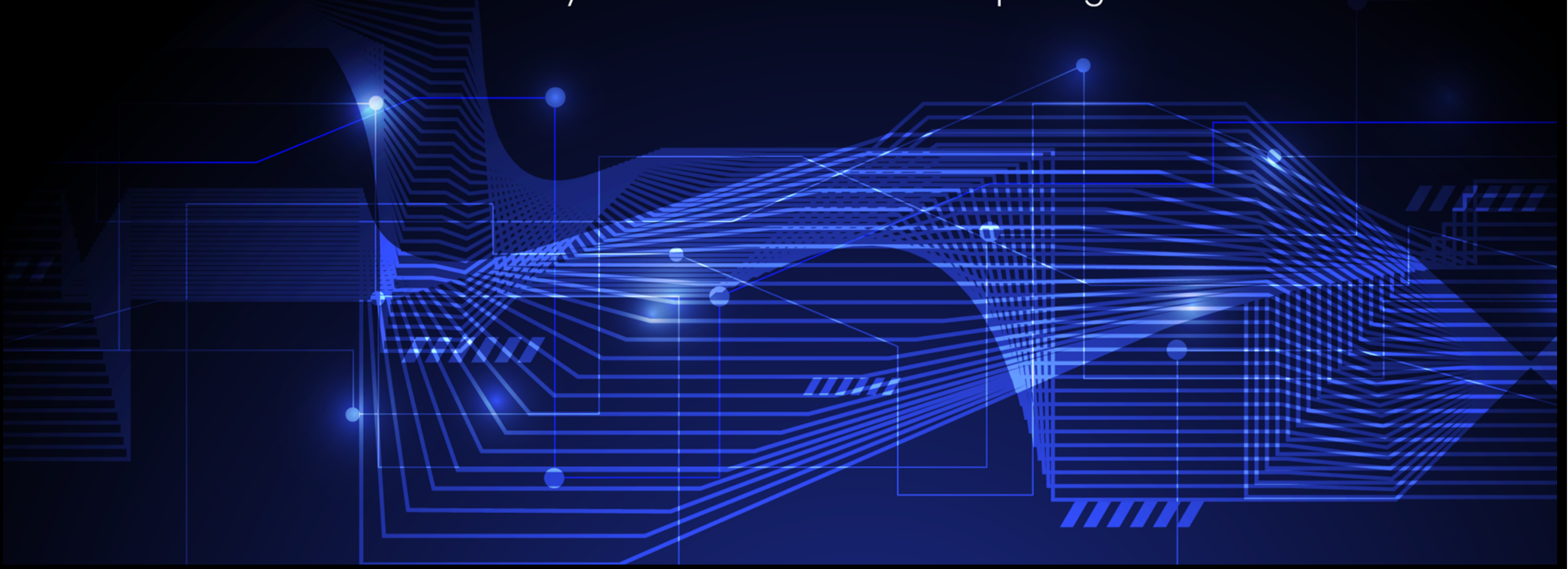
FairGate

The Key to Fair and Secure Computing



Fairgate White Paper

Introduction

Secure multiparty computation (MPC) is a subfield of cryptography that enables multiple parties to compute a function on their private inputs without revealing anything about such inputs beyond what can be inferred from the function's output. This is achieved through the use of cryptographic protocols that ensure the privacy and security of each party's input. Recent advances in MPC have focused on improving efficiency and scalability, as well as expanding the set of functions that can be securely computed. Fairgate's flagship technology is the FairRISCV processor. FairRISCV is a RISC-V processor that can run off-the-shelf binaries efficiently in a Two-Party MPC protocol with semi-honest guarantees. The FairRISCV processor not only can run finite programs that compute math functions, such as performing secret vote counting or deciding on Vickrey auctions, but can also execute continuous programs that produce many intermediate outputs, such as distributed consensus systems requiring private inputs (i.e fair ordering of transactions).

By leveraging a standard CPU core implemented as an MPC-circuit, we can run a variety of protocols using a proven toolchain. Some examples are fair trades using smart-contracts, privacy preserving retrieval of information, private AI training and model evaluation, private credentials, decentralized identity, and more. The use cases for multiparty computation are endless, and MPC is expected to grow in the forthcoming years. A recent public comment by Dan Boneh sheds light on what's coming:

> "*Right now the blockchain is focused on getting snarks to work fast. Don't be surprised that if in a year or two all of a sudden the whole blockchain ecosystem is going to focus on MPC and we're going to see massive massive investments in getting MPC to run on massive computations of this style for these kind of problems*"

Gartner also foresees the imminent need of MPC in large corporations:

> "*..By 2025, 60% of large organizations will use at least one PEC (privacy-enhancing computation) technique in analytics, business intelligence and/or cloud computing..*" -Gartner Strategic Planning Assumption [1]

This forecast has been growing, from a 50% predicted in 2020 [2], to 60%.

There are many problems where the FairRISCV can be a key part of an innovative solution, these include:

Secure data analysis: In scenarios where multiple parties have sensitive data that needs to be analyzed, the FairRISCV processor can be used to perform the analysis without revealing the underlying data to any party. This allows for secure collaboration on data analysis tasks, such as in medical research or financial analysis.

Fraud detection: The FairRISCV processor can be used to detect fraudulent behavior by combining information from multiple parties without revealing sensitive data. For example, credit card companies can partner with banks or merchants to execute common programs to identify fraudulent transactions by analyzing patterns across multiple databases.

Privacy-preserving machine learning: The FairRISCV processor can be used to train machine learning models on sensitive data without revealing the underlying data to any party. This allows for privacy-preserving machine learning, which is important in applications such as healthcare and finance.

Auctions: The FairRISCV processor can be used to enable secure and fair auctions without revealing sensitive bidding information. This allows for trust in online auctions, where bidders are not known to each other.

Supply chain management: The FairRISCV processor can be used to enable secure collaboration between multiple parties in a supply chain, without revealing sensitive data. For example, suppliers and manufacturers can jointly execute programs to collaborate on production schedules and inventory management, without revealing trade secrets or other confidential information.

Fairgate Technology Suite

Fairgate has specialized in solving the Fair Exchange Problem in numerous real-world use cases with its FairRISCV technology. A new FairTrade extension protocol allows users to atomically trade secrets with proven properties for cryptocurrency payments in a fully malicious-party protected setting. The secrets traded can be code vulnerabilities, MEV arbitrage opportunities, scientific discoveries, solutions to well-stated problems. The valuable properties of the secrets are tested by the FairRISCV program, so no action is performed unless the traded information is indeed valuable.

In the following sections we'll introduce the FairRISCV processor and the FairTrade protocol extension enabling a Fair Exchange of high-value assets. Fairgate is committed to use state-of-the-art advances on multiparty computation and zero knowledge systems, always with a pragmatic approach.

FairRISCV

To achieve fast program execution over MPC, we've developed an efficient RISC-V processor based on Yao's protocol that uses logic gates. Our processor makes combined use of the state-of-the-art MPC optimization techniques such as FreeXOR, Half-Gates, Row-reduction and RAM consistency. Fairgate has a roadmap of optimizations for the FairRISCV processor to improve bandwidth usage and reduce communication latency to take the MPC from theory to practice. Our roadmap includes using mixed logic and algebraic systems, implementing a FairRISCV circuit compiler, a custom MPC-optimized MAC for encrypted-memory consistency, and more.

Main features of the FairRISCV processor:

- **Standard**. Executes off-the-shelf RISC-V binary code between two parties in a semi-honest setting.
- **Mobile-ready**. Runs on low-end devices such as mobile phones or standard laptops.
- **Tuneable**. Easily fine-tuning security thresholds for maximum performance in specific use-cases.
- **Logic-circuit optimized**. The processor is implemented on a boolean circuit that is efficiently executed using Yao's protocol and later improvements..
- **Parallelizable**. The FairRISCV processor can be parallelized to take advantage of multi-core CPUs, SIMD CPU-extensions, the AES-NI opcode or GPUs.
- **Memory-optimized**. Memory usage can also have a significant impact on performance. Data structures and algorithms are chosen to minimize memory usage and reduce the number of memory accesses.
- **Encrypted RAM**. Support for private and encrypted RAM memory spaces.
- **Instruction Packets**. By splitting the processor execution in constant-step circuit packages, we're able to provide execution building blocks on-demand, reducing network bandwidth utilization.
- **Parallel gable circuit generation**. The packaging of instructions in stand-alone circuits enables the generation of packets in parallel, and the preprocessing and caching of circuits for future use.
- **Private-free and Secret CPU Register modes**. The processor can work in the so-called privacy-free mode, where the prover knows all the processor registers contents, or in secret mode, where neither the prover nor the verifier know the register contents. The former mode produces circuits that are half as long as the later on average. The processor also supports the single-secret mode (explained in the FairTrade section), where the verifier hides a single secret, and this secret is only revealed as the last step of the circuit execution.
- **Multi-Core systems with inter-core communication channels**. Our architecture enables parallel execution of several FairRiscV cores while providing privacy-free or secret communication channels between the cores. This allows the simulation of multi-core systems or independent agents. In the context of blockchains, this enables the proving of properties of contracts running in separate blockchains, communicating through bridges.

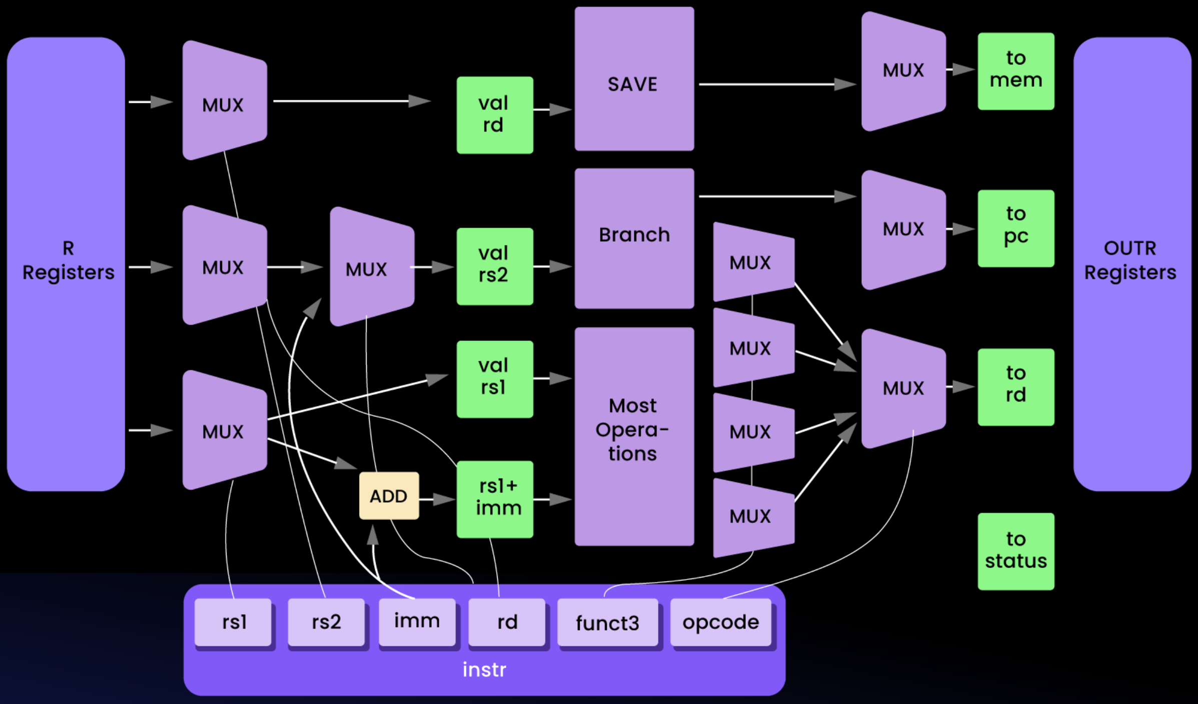
The FairRiscV processor supports the use of three concurrent address spaces:

- **Privacy-free RAM**: Both the addresses accessed by the opcodes and the contents of each memory cell are known to the prover at all times, but not to the verifier. This mode is ideal if the core is running in the **privacy-free** mode.
- **Encrypted-content RAM**: The content of each memory cell is unknown to both the prover and the verifier, but the addresses accessed by the code are known to the verifier (but not the prover). This mode is ideal if the core is running in secret mode, but the memory access pattern is predictable. It is common that certain algorithms can be implemented in a way that is address-pattern invariant. Sometimes cryptographic code is already address-pattern invariant when it needs to be protected from side-channels that exploit cache-misses, memory misalignment delays or electromagnetic emission patterns when accessing different RAM banks.
- **Encrypted-bus (oblivious) RAM** : Both the addresses accessed and the content of the RAM at those addresses is unknown to both parties.

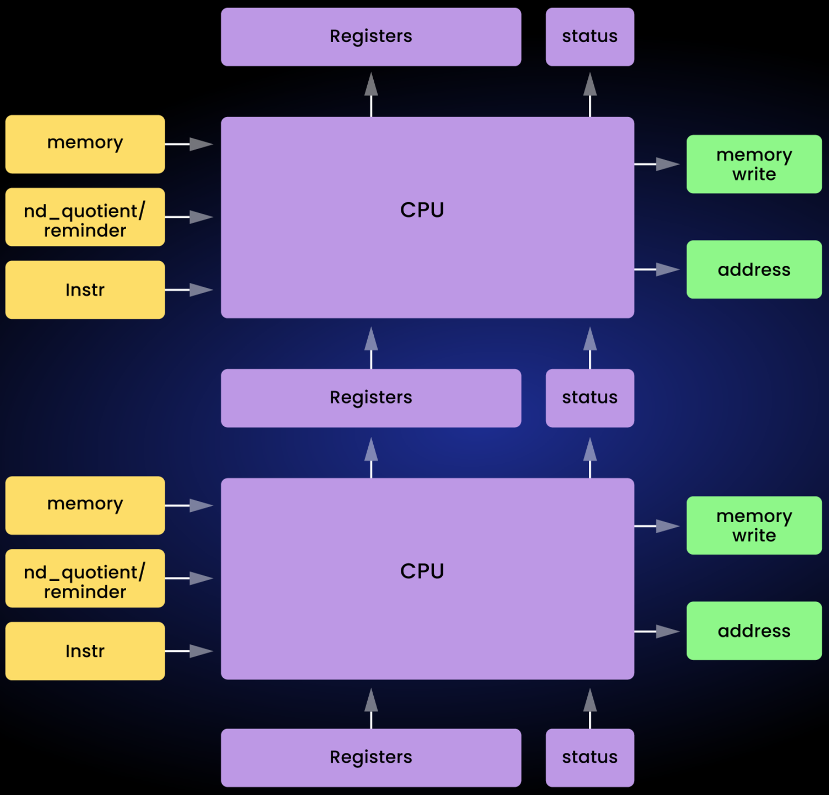
For each use case, the processor can be instantiated with one or several of these memory spaces. When using only private RAM, it is possible to instruct the processor to store certain secrets in processor registers, providing a fast encrypted-content RAM for short secrets.

CPU Design

The CPU consists of the R registers, input multiplexers, a simple ALU, the branch logic, the output multiplexers and the buses that connect outputs with memory, the program counter and back to the R registers.
The input multiplexer that choses a register based on the rs1 field of the opcode. All ALU circuits are executed, although short-circuit and lazy evaluation allows unused gates to remain unevaluated, reducing the CPU load. Output multiplexers choose which result must be written back to the R registers based on the opcode defined by the func3 field. Instructions are fetched from RAM, together with 2 other RAM arguments, corresponding to consecutive RAM words, which enables RAM access to unaligned words. Each instruction also produces an output that is either stored in a register, or stored back in RAM. To implement the integer division opcodes, an output witness is used, and the processor only needs to check for the correctness of the multiplication.



To execute multiple sequential instructions of the processor, the instructions are unrolled and each instruction gets a different garbled circuit. A sample execution transcript is shown in the following diagram.

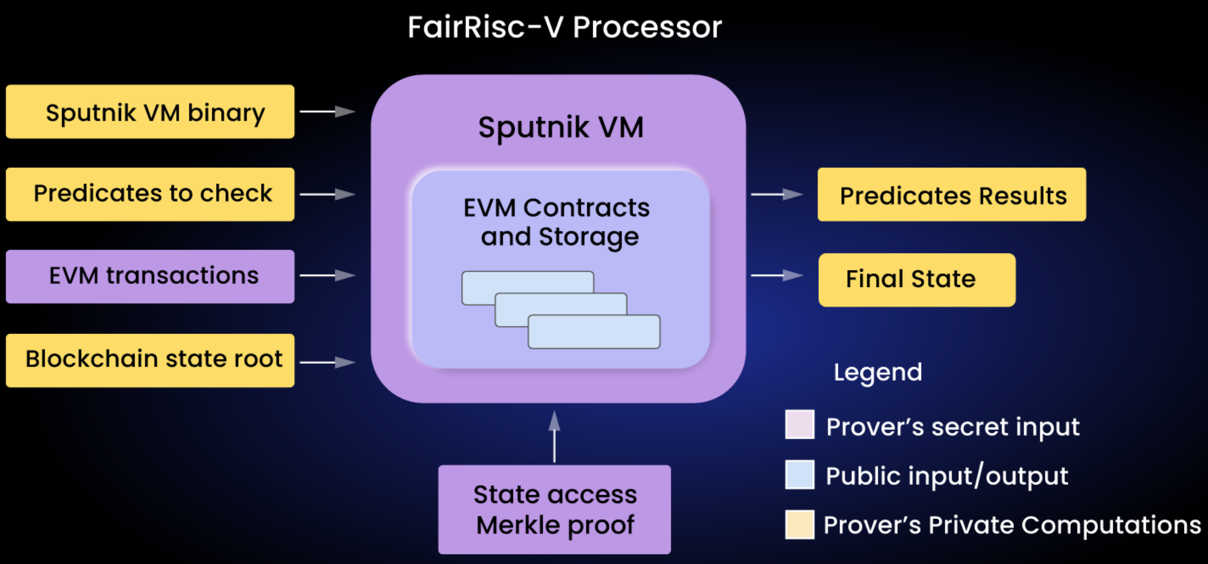


existing circuit formats such as Bristol or Fuse. When the processor is run in privacy-free or single-secret modes, the circuit file is augmented with meta-conditionals over wire states that allow skipping the evaluation of circuit components that are unused in a certain instruction cycle. The Fairgate circuit compiler performs a dependency analysis, reorder gates and inserts the conditionals in optimal positions to provide this functionality. The optimization reduces CPU usage, and could be paired with **stacked garbling** to also reduce bandwidth consumption.

We estimate that the performance of the FairRiscV can reach 1000 instructions/second when run in privacy-free or single-secret modes, on a standard laptop, using a single thread and taking advantage of the aforementioned optimizations.

FairRiscV running Virtual Machines

The FairRiscV processor can run other virtual machines by executing the VMs compiled code. For example, to execute EVM payload and check predicates about the EVM state in zero knowledge, we run a SputnikVM instance over the FairRiscV processor, and feed the SputnikVM with the blockchain state, a secret EVM transaction and other auxiliary secrets. This way we can prove properties of the transaction running on the simulated EVM.



The virtualization can also involve emulating third parties whose signing private keys are unknown to the prover, or contracts that have been tweaked to simulate uncommon or future scenarios. Using these tweaks, it's possible to emulate scenarios where the parties do not control all the keys, and third parties interact with the contracts in predefined ways, such as multi-party bridges.

Three methods are provided to this end:

- **ECRECOVER opcode selective-bypass**. The ECDSA EC public key recovery can be altered to recover certain public keys on predefined signatures. The selective-bypass mechanism can be applied under restrictive conditions (i.e. enabled in certain contracts, or at a specific instruction pointer).

- **State replacement**. The parties involved in the protocol can agree to replace certain parts of the state with data of their choosing. For example, multi-party wallets can be emulated so that the existing public keys are replaced with dummy publica keys. It also allows us to replace the code of specific contracts. For example, the owner of a contract can be modified, or the BLS public key stored in a contract that verifies BLS signatures can be tweaked to verify signatures for a simulated entity.

- **Third-party transaction injection**. The elliptic curve public key recovery method used to check the origin of a transaction can also be tweaked to enable the mapping of fake signatures into predefined account sources.

FairTrade

The FairTrade protocol enables two parties, Alice and Bob, to atomically exchange a secret (provided by Alice) for an on-chain cryptocurrency payment (provided by Bob) on a Smart contract platform such as Ethereum or Rootstock. The properties of the payment will be visible to Alice before the trade, while Bob's secret will only be revealed to Alice after the trade is successful. Accordingly, if the trade is successful the payment to Bob will be visible and transparent on the blockchain, while neither the secret nor the encrypted secret will, providing perfect secrecy.

The secret to be exchanged is verified so that the monetary transaction can only occur if the revealed secret has the desired properties. These properties are described by a simple algorithmic predicate. We give examples of useful predicates:

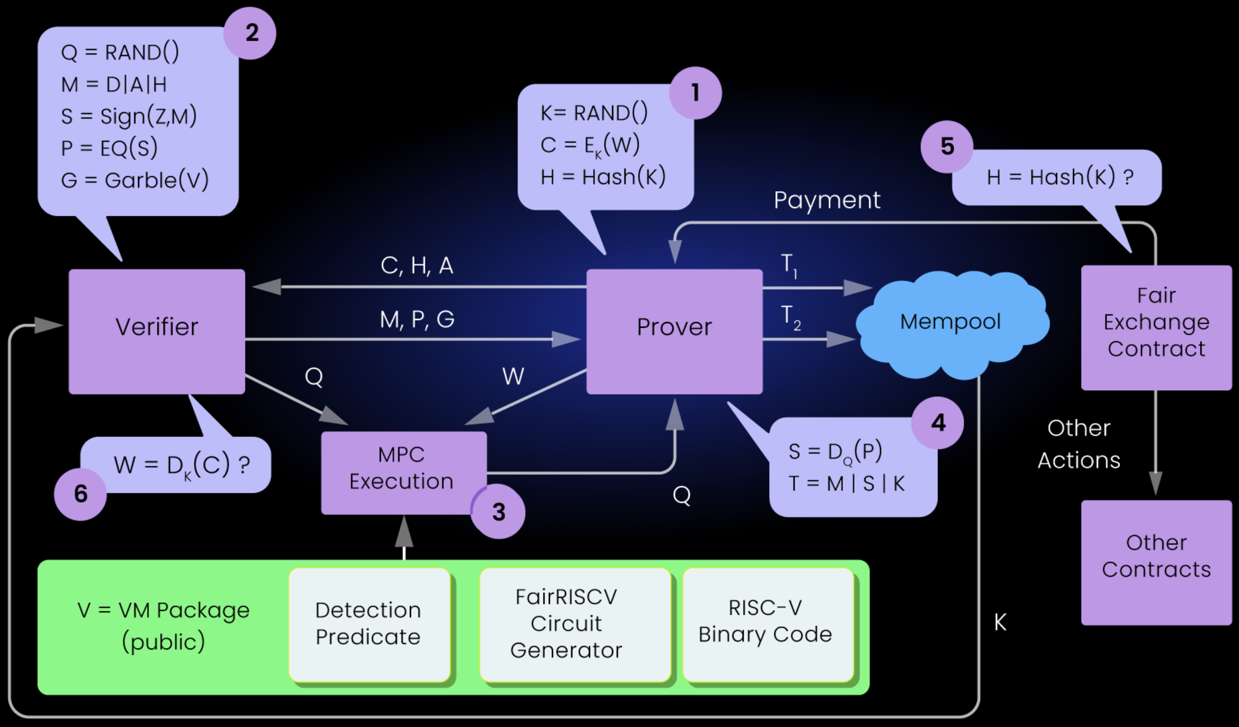
- Given a supposedly secure program, find a certain input that produces an anomalous end state. The prover can be paid a bug bounty for the discovery of that input.
- Given the state of several order books, detect a certain high revenue arbitration opportunity. The prover can be paid in proportion to the trade found.
- Given a certain AI neural network, detect an input that produces a dangerous, anomalous or biased response. The prover can be paid for searching and finding those edge cases.

To verify the predicate without accidentally revealing Alice's secret, the predicate is evaluated on the FairRISCV processor. Alice acts as the MPC prover and Bob as the verifier. To be able to prove the validity of the predicate efficiently, the FairTrade protocol runs the FairRiscV processor in semi-honest single-secret mode. In this mode, the processor is run in privacy-free mode, with the exception of the final gate computed, which reveals a single secret provided by the Verifier, and the revealed secret can be used by the prover to unlock the payment.

Main features of the FairTrade system:

- Full validation. Validate the quality and properties of the prover's secret information before trading.
- Variable payments. Pay for the information in cryptocurrency (including stablecoins, or any kind of digital asset controlled by smart contracts). Payment amount and related options can be algorithmically selected. For example, the payment amount can depend on secret information properties.
- Fast. Fair exchange can be accomplished in minutes, not hours, which makes it ideal for time-sensitive trades
- Privacy-friendly.The FairTrade system is ideal for non-delegable proofs, such as sensitive multi-million dollar vulnerability exploits.
- Control. Pair the fair trade with any on-chain action, such as executing specific smart-contract method calls.

The following diagram depicts the FairTrade process:



The Protocol is defined as follows:

Notation

- $E_K(M)$ a block cipher encryption function, and let $D_K(M)$ be the associated decryption function.
- Hash(M) a cryptographically secure Hash function
- Sign(M,K) a digital signature of message M with private key K.
- Garble(V) is an algorithm to encrypt the circuit V suitable to be executed in the MPC protocol. The encryption uses a one-time private key (omitted for clarity).

Participants

- Alice is the Verifier. Alice creates the garbled circuit.
- Bob is the Prover, receives Garble(V) from Alice and executes the circuit.

Initial Setup

1. The Verifier (Bob) locks funds to be available for trading in a Fair Exchange smart Contract (FEC). The verifier, who owns a private key Z, registers his public key Z_{pub} in the FEC, to be used later for signature verification.
2. The verifier makes the FEC address public
3. The Verifier generally establishes first the detection predicate, which is also made public. The Detection Predicate establishes the conditions in which the FEC would pay the locked amount to anyone acting as Alice.
4. The Verifier creates and publishes the VM public package, which contains the FairRiscV processor circuit generator and the program that the processor will execute, together with the detection predicate. The Verifier can offer several detection predicates (associated with different payment amounts).

Trade Protocol

Step 1

1. The Prover (Alice) poses a secret W that has some specific properties that can be verified in the MPC process.
2. The Prover creates a random one-time key K.
3. The Prover encrypt the secret W with the one-time key K, creating C = $E_K(T)$
4. The Prover hashes the key K, creating H =H(K)
5. The Prover chooses an on-chain payment address A to receive the payment in exchange for the secret.
6. The Prover shares C (the encrypted secret), H (a commitment to the key) and A (the address) with the Prover.

Step 2

1. The Verifier chooses a one-time random key Q that will be used for symmetric encryption.
   The Verifier chooses the Payment command identifier D, and associates it with the hashed key H and Alice's address A, forming a payment command message M. The payment command message instructs the FEC to perform the payment. The fields H and A will be part of the command arguments. The condition for the payment is that the preimage of H is algo given. For example, the full message M can be an **EIP-712**-compatible message such as performPaymentGivenPreimage(A,H). We do not restrict the message format here, and we simply specify it M = D | A | H. Note that the message M cannot be an EVM transaction with a Solidity ABI-encoded method call since an EVM transaction could be easily front-runned by Bob by reusing the nonce, canceling the payment. Note that M is an unsigned message, so it is useless without Bob's signature.
3. The Verifier computes a signature on the message M with his private key Z, as S = Sign(Z,M). This signature S, together with the message M and a preimage of H will be enough to trigger the payment.
4. The Verifier computes an encrypted signature P, as P = E_Q(S).
5. The Verifier (Bob) generates G = Garble(V), an encrypted version of the FairRISCV processor circuit. The garbled circuit will contain an encrypted copy of Bob's secret key Q.
6. The Verifier shares M (the payment command), P (the encrypted signature) and G (the garbled circuit) with the Prover.

Step 3

Now both the Prover and the Verifier are ready to perform the MPC.

1. The Verifier provides his secret input W to the circuit by performing an oblivious-transfer with the Prover.
2. The Prover provides his secret input Q by embedding it directly into the garbled circuit G, so this does not require further actions.
3. The Prover evaluates the garbled circuit privately and, if his secret W possesses the conditions specified by the detection predicate, he obtains Q

Step 4

1. The Verifier is able to recover the signature S by performing S = D_Q(P)
2. If the signature is invalid, it aborts the protocol.
3. The Verifier builds a message T_1 containing Bob's message M, S (the signature for M). We set T_1 = M|S (we leave the actual payload format unspecified)
4. The Veirifer sends T_1 as a payload in an on-chain transaction having the FEC address as recipient.
5. The FEC contract receives the message, and verifies the signature, and locks the funds to be paid to the verifier for a period.
6. The verifier waits until transaction T_1 has many confirmations.

7. The Verifier builds T_2 containing K (the preimage that enables the command). we note it simply T_2 = K.
8. The Verifier sends T_2 to the FEC.

Step 5

1. The FEC receives T_1 and unpacks all its fields: M, D, A, H, S.
2. The FEC verifies the signature S over the message M.
3. The FEC locks the funds for the payment for a period of time. No other party is able to propose a T_1 or T_2 message during this period.
4. The FEC receives T_2 from the same party and unpacks K.
5. The FEC verifies that H == Hash(K). If not, then it aborts.
6. The FEC unlocks the funds and performs the payment according to the command D, to the address A.
7. Optionally, the FEC can perform other on-chain actions specified by D.

Step 6

1. The Prover learns K from the EVM transaction while it is on the memory pool or when it is executed on-chain.
2. The Prover contains the secret W by decrypting C with the key K, performing W= D_K(C)

Case studies

The applications of the FairRISCV processor and the FairTrade protocol are endless. We’ve selected two initial problems in billion dollar markets that lack satisfactory solutions where the FairRISCV/FairTrade suite of products is a great fit.

Vulnerability Vault

The decentralized finance (DeFi) ecosystem has seen an explosion in growth and interest in recent years, with developers and users flocking to the technology to take advantage of its promise of trustless and automated financial services. However, the rapid growth of the DeFi space has also put it at risk of security vulnerabilities, with millions of dollars of users' funds being stolen in the past few years.

DeFi applications and smart contracts are vulnerable to security exploits due to the nature of their underlying blockchain technology, as well as a lack of proper security measures. In particular, the combination of complex smart contract code and the decentralized nature of the blockchain makes it difficult to identify and address security flaws before they are exploited. Additionally, the speed of the development process in the DeFi space often results in security vulnerabilities being overlooked or underestimated.

Bug bounty programs are a popular method for identifying security vulnerabilities in DeFi applications and smart contracts, as they allow users and developers to report security issues in return for monetary rewards. While these programs can be effective in finding and fixing security vulnerabilities, there is often an issue with the amount of reward offered for successful disclosures. Recent research shows that, for the vast majority of value locked in DeFi protocols, the bug bounty rewards offered are too low to properly incentivize responsible disclosure. There are several factors that contribute to the current alarming situation:

- Financial attacks (i.e. price manipulation using flash loans) are considered a legal gray area by some security researchers, who lack any experience in financial markets regulation.
- Bug exploits are also considered a gray area by “code is law” maximalists.
- Due to the ease to get hold of misappropriated digital assets, legitimate security researchers are incentivized to turn into malicious actors
- The decentralized nature of the blockchain means that malicious actors can often steal crypto assets, move them quickly across protocols and bridges, mix them with legitimate assets, and cash out while remaining undetected. Therefore these actors are not well incentivized to consider the bounty as a rational choice to avoid the legal risks of stealing and the cost of money laundering.

Additionally, communications between the security researchers and the protocol development team are often distrustful. The researcher doesn’t want to disclose all the information until having an assurance that a fair bug bounty will be paid for his work, and the development team doesn’t want to commit to a specific amount without fully understanding the severity of the vulnerability reported. While slow negotiations take place, the smart contracts are still vulnerable and other malicious parties can steal the funds at risk. This is a specially high risk for DAOs controlling smart contracts as pseudonymous insiders can access the vulnerability report before the contract code is patched and steal the crypto assets front-running the development team.

Overall, the security vulnerability landscape in the DeFi ecosystem is a serious problem that cannot be solved by existing security solutions alone. Bug bounty programs should be encouraged and properly incentivized in order to ensure that legitimate security researchers are properly rewarded for their work.

Fairgate Vulnerability Vault (FVV) is a native solution to handle the reporting and upgrading of smart contract vulnerabilities while protecting the funds exposed and securing an adequate reward to the researcher who found the problem.

VV significantly increases the trust between the security researcher and the DeFi protocol development team by providing a methodology to follow and the tools they need to successfully satisfy all the parties involved.
The FVV product bases its security in the FairTrade protocol and the FairRiscV processor. The researcher takes the role of the FairTrade prover, while the DeFi development team takes the role of the verifier. Together, they achieve a peer to peer secure simultaneous exchange of secrets. The predicate checked in the FairTrade protocol is the condition that the safety of the funds in the DeFi contract has been broken. For example, the predicate can check if the balance of all user's deposits does not match the actual balance of the DeFi contract. The predicate may also detect state incongruences, or highly outliers in oracle prices, and provide different bounty amounts in each case. The FVV protocol allows the following operations occur atomically:

- The researcher discloses the vulnerability to the development team
- The development team pays a fair bug bounty to the researcher
- The vulnerable smart contracts are automatically paused.

The fact that the smart contracts are automatically paused prevents the race-condition where a malicious researcher receives the bounty but immediately sells the vulnerability in the black market, making the highest damage to the protocol without personal risks. It also prevents the problem of inside jobs by pseudonymous remote code developers that often participate in DeFi teams. This is specially important for DAOs who may become legally liable for anything happening with the funds at risk after a vulnerability report.

The FVV is an instantiation of the FairTrade protocol, so it's a two-stage protocol. After the first stage is completed, the researcher receives a secret key that can be used to trigger the atomic exchange of secrets on chain. Also, even before the on-chain part of the protocol, the researcher is able to convince the development team of the existence of the vulnerability by showing a cryptographic hash of the key.

After the smart-contract fix is ready, the funds can be transferred back to the upgraded contract.

To summarize, FVV provides:

- A set of smart-contracts to facilitate secure and responsible vulnerability disclosure and bounty payments.
- An open-source toolbox for Security Researchers to privately simulate and then handle the vulnerability disclosure process securely and atomically
- Consulting services for companies that wish to define specific vulnerability predicates to narrow or adapt the scope of bug bounty conditions.
- A legal framework to protect the researcher and the DeFi development team
- Technical advise on the vulnerability fix and its deployment

MEV Matcher

Blockchains work by creating blocks, which are published by proposers and consist of a list of user-generated transactions that are executed in order. However, the ability of proposers to change the order, censor or selectively add transactions has led to the rise of a MEV extraction industry. In this industry, users pay money to searchers, builders and proposers who specialize in extracting value, building profitable blocks and earning fees for block inclusion. Flashbots aims to reshape this value extraction industry such that it benefits users through chargebacks or revenue sharing. One particular link in the flashbots value-extraction chain is the interaction between the user and the searcher. The user wants to keep their transaction confidential from the searcher, usually a DEX trade, and also the searcher wants to keep their arbitrage search strategy confidential from the user as well as the builder. One particular strategy is backrunning, which allows for efficient arbitrage and liquidations after a user transaction without negatively impacting the user. Both the user and the searcher can profit from creating a backrunning transaction. This win-win situation calls for a protocol that can ensure confidentiality until a backrunning extraction opportunity is found and the revenue is fairly shared between the user and the searcher. The problem has been clearly presented in the recent Flashbots article. The article presents a simplified use case with only these two parties, and restricted trade and arbitrage primitives. We tackle the problem in a more realistic scenario and we are able to prove arbitrage and reveal a key that enables the user to claim for a reimbursement. A more advanced setup could enable the secret encryption of the user's transaction and the backrunning transaction to be delivered to the bundler, where the encryption is a hybrid public-key, symmetric key scheme.

The FairRiscV processor together with the FairTrade protocol can provide a solution to this particular problem with a realistic resource footprint. To make the solution efficient without requiring more expensive MPC protocols, we run our FairRiscV processor twice, using opposite prover-verification roles, and we use a commit-reveal scheme to compare results before revealing any output to the other party. Since the user reveals his commitment first, we tolerate a 1-bit selective abort leakage of the user transaction (which is ephemeral), while we do not leak anything from the searcher strategy. The 1-bit leakage means that a malicious searcher may learn, for example, if a certain token was bought, but neither in which DEX pool, nor the amount. A malicious searcher can potentially learn the answer of a single yes/no question.

To perform public key and symmetric encryption with the FairRiscV processor we can either use the Risc V internal registers as temporary storage (running the processor in secret mode) or we can use an external encrypted RAM. The ephemeral key required for

ECIES encryption is created using a seed built from secret entropy in inputs provided by the parties.

Common knowledge:

- The bundler 's public key.

User secret inputs:

- Entropy E_U
- The target transaction T.

Searcher secret inputs:

- Entropy E_S
- A program that codifies a strategy for searching for arbitrage opportunities.

MPC Processing:

- Execute user's transaction on supplied state
- Search for arbitrage opportunity
- Compute cashback based on revenue-sharing contract.
- Creation of an arbitrage transaction with cashback.
- ECIES-Encrypt both transactions with the bundler's public key.

Execution Output:

- ECIES-Encrypted transactions.

Multiparty Cryptocurrency Custody

MPC (multi-party computation) is considered the **next generation** of private key security. The FairRiscV processor can provide multi-party signing of ECDSA, EdDSA, Schnorr, BLS or even post-quantum signatures by removing the concept of a single private key; such a key is never gathered as a whole, neither during the first creation of the wallet nor during the actual signature. We developed a FairRiscV two-party protocol where both participants can enforce security policies based on amounts transacted, destination addresses, rate limits, contract calls, allowances and tokens accessed. In practice, one endpoint is a custodian, while the other endpoint is the user's wallet, each one holding only a share of the private key.-
During preparation, the endpoints engage in a decentralized wallet creation protocol in which they compute the public key (wallet address) that corresponds to the set of individual private shares. When a signature on a blockchain transaction is requested both endpoints engage in a distributed signature process where each endpoint individually validates the transaction request and the spending policy and together they cooperate to sign the transaction.

[1]
https://www.gartner.com/en/newsroom/press-releases/2022-05-31-gartner-identifies-top-five-trends-in-privacy-through-2024

[2]
https://www.gartner.com/en/newsroom/press-releases/2020-10-19-gartner-identifies-the-top-strategic-technology-trends-for-2021