Summary

This paper describes the purpose and architecture of ADAPT, a developer framework for building distributed data meshes that underlie a broad range of Internet applications and replace traditional monolithic web back ends.

Monolithic Internet application back ends are not only a security liability in that a compromise threatens integrity and confidentiality of **all** user data managed by the application, often affecting millions or even billions of users. They are also a driving force behind abuses perpetrated by the application providers themselves, such as aiding authorities in totalitarian countries and training AI models on proprietary data. When asked whether they are able to build applications in such a way that user data is inaccessible to them, application providers, correctly, answer that, no, there are no tools that would currently enable them to do so at scale.

ADAPT is just such a tool, it:

1. enables development of distributed data and processing node meshes
2. simplifies use of advanced security and cryptography primitives for programmers with no specialized knowledge of code hardening
3. ensures a safe and healthy software life-cycle in distributed node networks
4. enables trustworthy code execution even if code is running on infrastructure belonging to someone else

ADAPT is not a blockchain nor is it targeting market segments normally served by blockchain. Instead, ADAPT adopts principles of decentralization towards non-financial use cases and, consequently, operates on an entirely different set of principles. Its main goal: to reduce reliance on social guarantees by the software service providers ("human trust") in favor of what we call "technologically-assured trust".

1. Background

Software service providers have absolute control over the software and the data pertaining to its users. Oft advertised controls, "Chinese walls", and any extant mechanisms ostensibly preventing access by employees to sensitive user data are completely subordinate to social circumstances, such as political pressures and financial incentives. Nobody likes this state of affairs, but everyone tolerates it, because we currently have no other way to build software. And once the unlimited agency of application's operators is impossible to eliminate, it is also impossible to protect the data from these pressures.

the degree to which data misuse affects everyday life of ordinary people is ever-increasing

At the same time, the degree to which data misuse affects everyday life of ordinary people is ever-increasing. Whereas in 1995 a hack of a website would be considered a non-event, today, families can literally starve to death were they to be blocked from access to some online services, such as banking, or government databases.

Despite the ingrained monopolies of Google and others, there is a significant potential for disruption in Internet software market sectors in which the users exhibit high degree of dependency on the software providers. In some cases users can no longer rely on Internet software for their activities, as seen from the increasing unwillingness of enterprise to let their employees use SaaS services for fear of IP misuse. The disruption will come from services that completely eliminate their own ability to affect the system's operations and to access or affect the data within the application. But this reduction of agency has to be architectural and technological, not legal or reputational.

disruption will come from services that completely eliminate their own ability to access the data

At this point in the history of computing (2023) we have working examples of systems in which the agency of operators is significantly diminished in order to increase security and reduce incentives for abuse. These come in the form of blockchain platforms. Blockchains are globally synchronized financial data stores with technologically-assured guarantees of consistency (a.k.a. the double-spend problem) and, sometimes, availability (censorship-resistance).

blockchain systems lack in many respects

However, blockchain systems lack in many respects. First, they are expensive (this is the case even with the second and third generation platforms), because global synchronization of a large data set has a specific bottom price. Second, assurances of global consistency and correct code execution are insufficient for some use cases, and are unnecessary for others. Third, blockchain systems and virtually all of the components of the blockchain software stack are highly sensitive to upgrade risks, which prevents a healthy evolution of such software. In the end, blockchain systems are useful to some niche forms of financial innovation and not much else.

blockchain systems ... are highly sensitive to upgrade risks

Within SaaS, finance, and IoT industries there remains a strong need for systems built with architecturally-assured guarantees of integrity and confidentiality, something that blockchain systems do not deliver. This trend is exemplified by industry-wide calls for adopting zero-trust architectures; cloudless IoT authentication that provides users with self-custody of their authentication credentials; and privacy-preserving KYC, which aims to establish privacy guarantees while still enabling meaningful regulation and surveillance.

Despite web3's forward-looking alignment with the principle of reducing operator's agency, the tools emerging from the web3 space have seen little to no adoption in non-web3 use cases. To close the market gap between web3 technologies and enterprise, several problems need to be addressed first:

- Blockchain is a "fifth wheel", when it comes to enterprise systems. Blockchain technology mainly focuses on availability and consistency guarantees, which are rarely part of the enterprise software core threat model.

- Web3's emphasis on payments and (crypto-)currency, and, in general, on settle-able financial assets as its core focus misses enterprise customers that see the current payment rails as perfectly suitable to their purpose.
- Web3's lack of focus on privacy, or focus exclusively on privacy of payments but not other types of confidentiality, is unsuitable for enterprise, with its view of privacy and confidentiality as a core requirement.
- Finally, web3 software is immature from the standpoint of enterprise developers. This is not so much the question of code quality or reliability, but more so of appropriate modularization and long-term maintainability that constitute a difficult socio-engineering problem in decentralized software in general.

ADAPT aims to address these issues, both extending and hardening web3 technologies beyond blockchain and enabling enterprises to deliver systems with security guarantees appropriate to the ever-evolving threat landscape.

2. ADAPT Framework

cryptographic primitives are critical building blocks

Web3/blockchain frameworks have demonstrated use of cryptographic primitives such as hash commitments, cryptographic signatures, and secure multi-party computation as critical building blocks for software in which the operator agency is reduced. Other innovations in cryptography, such as ZKPs, are rapidly gaining momentum and applicability in this space. Consequently, we see these building blocks as essential for the new generation of disruptive Internet software systems.

ADAPT addresses critical developer needs when it comes to building cryptography-heavy software

The difficulty with extensive use of cryptography is its inherent fragility. Not only is it hard to use advanced cryptography correctly (even developers that utilize third-party cryptography components need specialized skills), but, more importantly, the primitives themselves are fragile to later discoveries of critical exploits, the impending quantum computing revolution, and the steady increase in computing resources dedicated to brute-force attacks. There is also fragility such systems display in long-term maintenance: software upgrades are both difficult and dangerous in the decentralized context, because the upgrade process breaks the very guarantees that the systems are designed to provide.

ADAPT provides a specialized virtual machine – a cryptographic data container

ADAPT vastly simplifies the process of building cryptography-heavy software by providing a specialized virtual machine – a cryptographic data container. The data container is, internally, a specialized computation environment, and externally – a single component dedicated to (and only to) storing and processing any data to which the desired security guarantees apply.

Data containers consist of data (represented in a high-level and generic data format similar to JSON, but with extra features, such as pervasive Merkle hashing), and of code (developed in a domain-specific language called MUFL) that oversees all business logic associated with the data stored within the container. Like data, code objects are content-hashed, enabling a variety of features related to remote software attestation and ZKPs.

ADAPT supports a range of protection mechanisms, including TEEs

Orthogonally to the evaluation and storage environment within data containers, the ADAPT framework also supports a range of protection mechanisms to prevent unauthorized access to the containers. To this end, the framework utilizes current industry-standard approaches: TEEs (for server side) and WASM (for client side). The system will support a ZKP-based security framework in the future.

ADAPT cryptographic data containers, in combination with security environments appropriate to each data node, form a secure data fabric that underlies each specific security-sensitive application.

ADAPT containers form a secure data fabric of an application

The goal of this architecture is to enable the framework to control critical aspects of the cryptography-heavy software system, which would otherwise be expensive to deliver and error-prone to build, such as memory-safe message parsing, deterministic computation, and portability (including to front-end environments). Delegating these tasks to the framework results in a significant improvement of long-term system security, while significantly reducing the cognitive burden placed on developers, especially in the context of long-term maintenance.

ADAPT evaluation environment

In this brief section we outline the ADAPT's data model and execution formalism.

A set of valid data values D in Adapt is a union of sets A, F, M, S, where: A is a set of atoms consisting of strings, integers, booleans and other primitive types; M is a set of key-value maps of the form $(D, D) \to D$; F is a set of functions $(S, D) \to (S, D)$; and S is a set of state snapshots explained below.

As the data container undergoes state transitions as a result of invoking one of its state transition functions with some input, a state snapshot represents one of its states. Formally, a snapshot contains a tuple of n values $d \in D$ and a tuple of m state transition functions from F called transactions, plus a special initialization function F_0. Note that because values from S are automatically included in D, state snapshots may include other state snapshots as nested data items (the usefullness of this will become apparent when we discuss using ADAPT in the context of ZKPs).

All data in ADAPT is hashable, that is for any $d \in D$ a content-hashing function $H(d)$ is defined and has "good" cryptographic properties. Because the sets F of functions, and S of snapshots are contained in D, any function or snapshot has a "good" content-hash identifier. Additionally, modules (collections of functions) in ADAPT are also content-hashed. ADAPT type system is shown in Figure 1.

ADAPT packets reside inside an execution environment, an evaluator. The evaluator hosts a dag or chain of packet's state snapshots, or the latest/current state. The evaluator accepts and processes requests sent to the packet by another network component. We will go into more detail about the types and functions of different evaluators in due course, but note briefly that the framework currently supports Linux-based server-side evaluation, Amazon Nitro secure enclave environment, and WASM-based front-end environment for browser and mobile components.

ADAPT includes a convention for how packets can cause requests to be sent to other packets. Such requests are sent by producing specially-formatted output as a return value of a state transition function. They are processed by the execution environment, which is tasked with using the network capabilities of the hardware to send them to the recipient via a message broker.

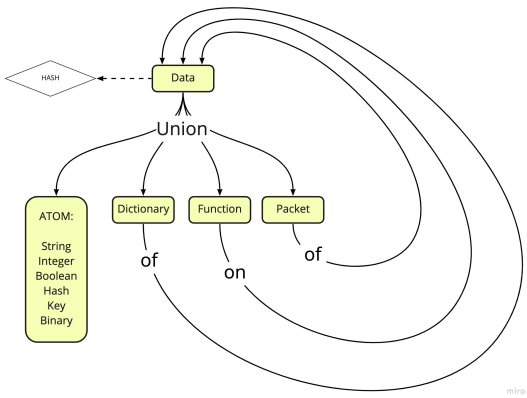

Figure 1: ADAPT type system

3. ADAPT State Transition Model

ADAPT's state transition model was developed so as to make ADAPT containers into suitable building blocks for distributed data meshes. The state transitions of an ADAPT packet are shown in Figure 2. They enable one to construct either a chain or a DAG of states easily, which leads to the simplicity of code used to answer such questions as "what's the longest chain", etc.

The intuition behind Figure 2 is that executing a transaction on an ADAPT container will transition the container from the initial state into the resulting state. Transactions take input and generate output.

More formally, given a state snapshot $s_i \in S$, input $d_i \in D$, and a transaction function $t \in F$ a state transition generates a new state s_o and some output $d_o \in D$. Note, that for transactions that retrieve data without modifying the packet, $s_i = s_o$. Recall that all of s_i, s_o, d_i, d_o, and t are content hashed.

It must be noted that use of cryptography sometimes requires use of randomly-generated data (entropy). Entropy comprises an additional input to transactions (not shown in Figure 2) provided by the security environment in which the container resides.

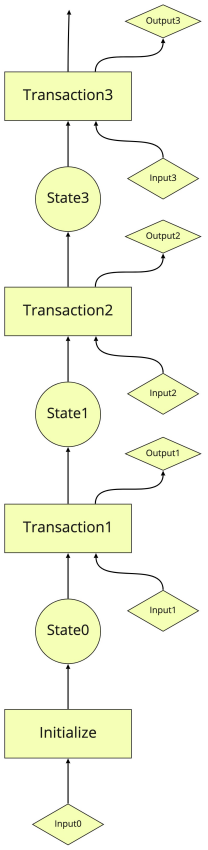

Figure 2: ADAPT packet state transitions

4. Messaging system

workflows are implemented as sequences of messages between containers

To assemble an application from ADAPT containers requires that certain business logic can involve data stored in multiple containers. We term such logic *secure workflows*. These workflows are implemented as sequences of messages exchanged between ADAPT containers.

The framework provides the necessary infrastructure. Messages can be emitted by containers as a result of a given state transition, or by the users of the application, as a result of them engaging with the application's UI. ADAPT includes a message broker component the goal of which is to forward messages between containers belonging to the same application.

The resulting layered structure of the ADAPT framework is shown in Figure 3.

Figure 3: Layers of the ADAPT architecture

messages are only sent to and received from properly verified components

Because secure workflows often require remote trust, the framework carries out all possible verifications at the messaging layer, ensuring that an invalid message does not reach the addressee container. This includes, for example, a situation in which one or both of the interacting nodes use a secure enclave environment for protection. In this case, the software attestation process provided by the enclave(s) is used by the messaging framework to ensure that messages are only sent to and received from properly verified components. This greatly improves programmability of software that makes use of secure enclaves. A similar mechanism will be provided for ZKP-based security, which similarly requires verifying the software being proven.

5. ADAPT as a smart contract framework

Similar to other innovations pertaining to decentralized software, the term smart contracts first came from web3, referring to code executed in an environment of high security for the purpose of financial transactions. That is, smart contract code execution environment (blockchain, in the case of web3 systems) is what enables this, because it provides technologically-assured (rather than social) guarantees of integrity, consistency and availability.

The term smart contract is a misnomer, taking its root from the fact that such code is most often used to mediate financial interactions which some may naively see as "contractual", e.g. payments made when certain conditions are true.

smart contracts are secure workflows over a common dataset

From the software engineering standpoint, *smart contracts are distributed secure workflows over a common dataset*. In the case of web3, this dataset often represents financial ownership data. However, the applicability of distributed secure workflows does not end there by a long shot.

even in simple non-financial systems, interactions between nodes can be viewed as smart contracts

In practice, any environment that aims to reduce operator agency must enable some form of smart contracts. Even in such relatively simple software systems as secure peer-to-peer chat, the interaction between two clients can be described as a smart contract. For example, using ephemeral messages to communicate sensitive information requires party A to trust that party B will delete the message at the pre-agreed time. The code that carries this out within B's chat client is a smart contract. This is where ADAPT's general view on decentralization departs from the convention web3 view, in which smart contracts are seen as a purely financial primitive.

ADAPT's data containers were designed to enable such general form of smart contracts. Their highly-portable deterministic execution environment and its approach to representing data and business logic as universally content-hashable can be used to implement financial primitives, but is also suitable for a much broader set of use cases.

verifying the security claims one system's node makes to another are essential for building robust systems

It is important to note that the security guarantees of smart contract code in ADAPT are not the property of the code itself or the execution framework, but rather of the environment into which the framework is embedded. TEEs and ZKPs provide a different set of security guarantees and answer to different threat models. Understanding the limitations of the secure environment, and, when possible, verifying the security claims one system's node makes to another, are essential for building robust systems.

To summarize, ADAPT implements a highly portable, security-sensitive, isolated, and distributed execution framework, whose sole purpose is to process requests that manipulate security-sensitive data, often using advanced cryptography primitives. In essence, ADAPT implements an abstract smart contract mechanism, which can then be embedded within any secure environment, without making almost any assumptions with respect to the specific use case in which it is used.

ADAPT implements an abstract smart contract mechanism

6. ADAPT for SaaS

the way SaaS systems store data leads to the so-called "honeypots"

Figure 4 shows a typical way SaaS applications are architected from the security standpoint. The point of the drawing is to show that the security of a SaaS software back-end relies on the security perimeter established by the organization that owns it. Client nodes are speaking to a black box that does not have a way to prove to them any facts about its security arrangement.

Furthermore, the way such systems store and organize data leads to the so-called "honeypots", locations that are constantly and indefinitely increasing the potential value that could be extracted with an exploit. Honeypots are not insecure by definition, but because they grow to attract ever-increasingly motivated attackers, they must also increase their overall security in step with their growth. This rarely happens in practice, since security is a cost center, not a profit center. With costs of security increasing non-linearly as the threat pressure grows, security of SaaS products inevitably falls behind the value of their data honeypot.

security of SaaS products inevitably falls behind the value of their data honeypots

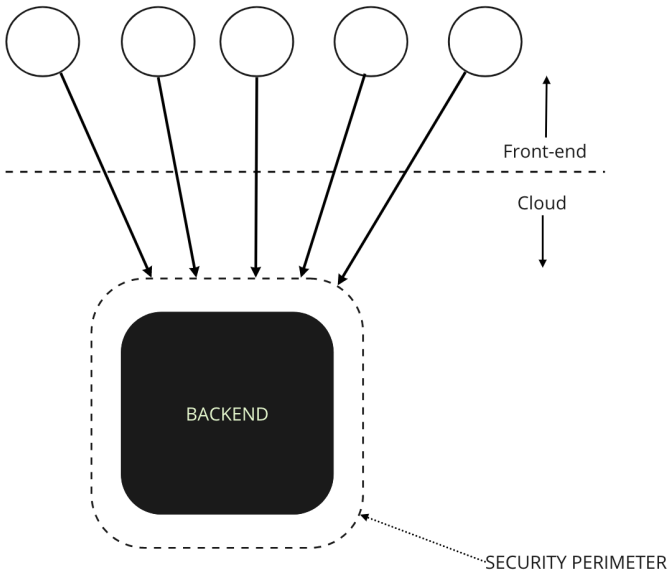

Figure 4: Typical simplified architecture of a SaaS system

the data honeypot is replaced with a mesh of nodes

The architecture made possible with ADAPT is shown in Figure 5. Here, the data honeypot that used to accumulate on the back-end of the SaaS application is replaced with a mesh of nodes, each storing data pertaining to its own needs, or a specific segment of the application. The security perimeter established by the organization owning the given SaaS service is removed, and the appropriate security mechanism is provided by each node, individually.

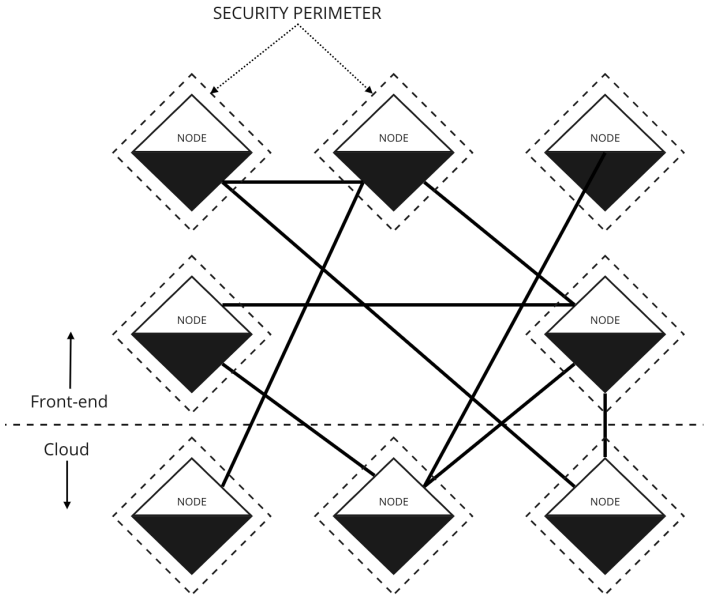

Figure 5: Peer-to-peer architecture with ADAPT

ADAPT can be used to create non-custodial applications

Benefits of ADAPT do not end with the breaking up the data honeypots and with providing easy programmability of secure workflows between individual nodes. ADAPT also delivers a mechanism to establish remote trust between nodes wherever possible, introducing security guarantees can be verified by anyone within the network, and so reducing the agency of the system's operators.

Systems that make this possible today are: secure enclaves and zero knowledge proofs. Adapt currently supports full scope security with enclaves, and the team is building a zero-knowledge proof system appropriate for the ADAPT environment. Weaker security environments of mobile and front-end nodes are also supported via WASM.

ADAPT vastly improves on the SaaS service architecture by breaking up the back-end data store, and enabling remote trust with client-side verification. The system can be used to create non-custodial applications, improving the degree of protection users have from both exploits and data misuse.

7. ADAPT for Web3

ADAPT targets systems irrespective of the kind of security they require

Blockchain has significant limitations stemming from its core focus on global consistency and availability (censorship-resistance). This means that blockchain fails as a platform for use-cases that require privacy (data sharing, key management) and is needlessly expensive to employ for use cases that require trustworthy code execution, but not global consistency, such as derivatives finance. Figure 6 illustrates that nodes interacting with blockchain, are not architecturally-trustworthy, despite claims of radical decentralization made by blockchain developers.

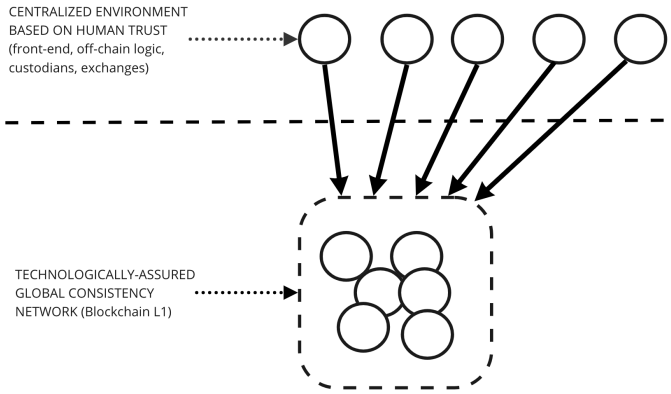

Figure 6: Human-based trust still present in blockchain-based systems.

ADAPT expressly targets systems irrespective of the kind of security they require. The nodes shown in Figure 5 can be globally synchronized (after all, achieving global consensus is just another secure workflow between system's node). But since ADAPT supports other types of security, it becomes broader than blockchain in its ability to meaningfully support non-financial use cases that rely on securing critical data, even from the operators themselves.

8. Zero-knowledge environment

ZKPs provide trust of remote computation

Zero-knowledge proofs (ZKPs) have recently gain notoriety in the blockchain/web3 space. In short, ZKPs enable a software node to convince another node that some computation has been carried out faithfully, while, optionally, hiding the inputs of the computation. This has many applications, the main of which is the ability to enable trustworthy remote execution of code in use cases which used to require human-based trust.

A good case study in this area is peer-to-peer gambling, such as poker. ZKPs enable a provably-fair construction of common source of randomness for participants, which is an incredibly difficult problem if tacked without them. Notably, much like with derivative finance, p2p gambling only requires blockchain for settlement. The game-play itself does not require global consensus, merely the local agreement between the playing participants.

The basic building block upon which we will heavily rely is the zk-proof associated with state transitions of an ADAPT packet from Figure 2. Given a state snapshot $s_i \in S$, input $d_i \in D$, and a transaction function $t \in F$ a state transition generates a new state s_o and some output $d_o \in D$. Recall that all of s_i, s_o, d_i, d_o, and t are content hashed. A zk-proof can be generated, which, given hashes $H(s_i)$, $H(s_o)$, $H(d_i)$, $H(d_o)$, will attest to the validity of the state transition.

This mechanism enables concise programming of peer-to-peer financial contracts

We can chain these together, providing a proof for a sequence of state transitions. We can further generalise this as follows: given a packet ID and a pair of state hashes (where the origin state may be optionally NIL for a proof that includes packet initialization) we can generate a proof of validity of a subsequent state hash of the packet after carrying out N transactions. We can include a single composite hash that merkelizes the entire set of interim states, inputs, and outputs. Using the composite hash, we can now selectively reveal any facts about the packet or any of its interim states or transactions. A more detailed drawing of the generalized state proof is shown in Figure 7.

This mechanism enables remarkably concise programming of peer-to-peer contracts of the sort illustrated by the gambling use case. For example, the example code that constructs a provably-fair shared card deck between two players is about 100 lines of code, of which the majority is simply constructing a private card transposition, and only about 20 lines are dedicated to ZKPs and hash commitments required for the process.

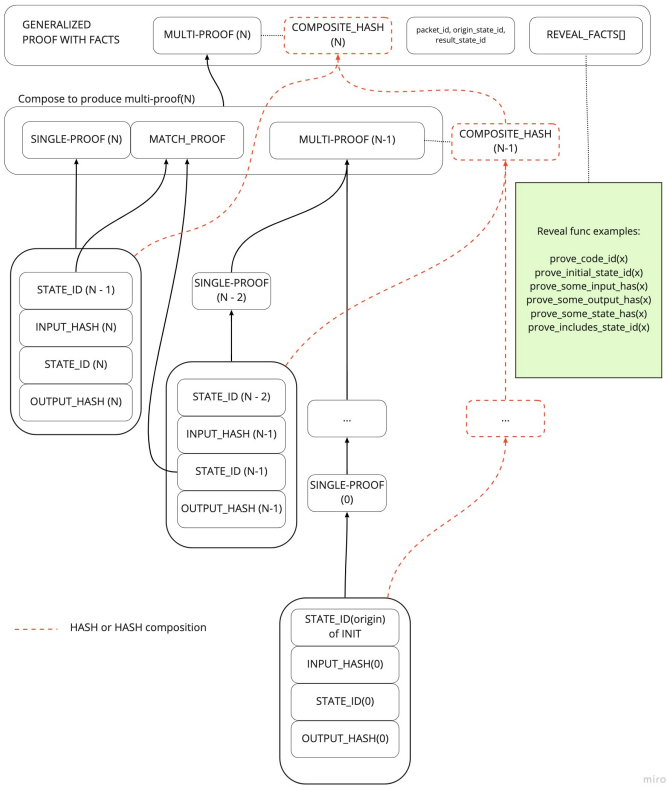

Figure 7: Inductive generalized state transition proof

Generalizing state transition proofs

Figure 7 shows the inductive mechanism to form a generalized state proof $GP(m,n)$, given $GP(m,n-1)$. $GP(m,n)$ consists of a tuple of the following values:

1. a multi-proof $MP(m,n)$, which is a composition of zero-knowledge proofs $P(n)$ of the correctness of the state transition n, a multi-proof $MP(m,n-1)$, and a proof-of-match, $PM(n)$, which proves that the initial state hash at state n matches the result state hash at state $n-1$;
2. a composite hash $CH(m,n)$, which is a merkle-composite of $CH(m,n-1)$ with the data tuple associated with $P(n)$, which includes the state hashes s_{n-1} and s_n, and the input d_i^n and output d_o^n of transaction n
3. a reveal and correctness proof of values associated with the transition from state m to state n: the ID of the packet, and the hashes of the two states
4. and it can optionally include a list of revealed (and proven) facts associated with the state transitions or the packet, $R(m,n)$ which we will discuss below.

9. Fragility of remote trust security

remote trust guarantees that some remote computation was performed correctly

Trustworthy remote computation is one of the forms of technologically-assured trust. It guarantees that some computation performed by a different party was carried out correctly. The two technologies that enable remote trust of computation are TEEs and ZKPs. TEEs are hardware components that provide confidentiality and software attestation mechanisms to ensure that the computation is performed as expected. ZKPs enable party A to prove to party B that some computation was performed correctly using cryptographic means.

remote trust requires pinning down the version of the software

Both of these technologies (and all other approaches that achieve the same purpose) must pin down the version of the software being proven or attested. Trustworthy remote computation must assert that software being executed matches some known and expected version, down to each specific instruction. If they lacked this mechanism, such primitives would be entirely useless, since proving faithful execution of unknown software carries no benefits.

security is a process, not a state

However, no software can ever exist as a single version. Software undergoes fixes and improvements, and in any real live systems multiple versions must be supported. This is especially true of security-sensitive software, since it is unreasonable to expect that a specific software package contains no exploits.

any real live system must support multiple versions

Security, in other words, is a process, not a state. Software that wishes to remain secure in the long term must be easily upgradable. Yet, upgrading software in the presence of security primitives that pin down the software means introducing a choice of verifiers, one for each valid version. To eliminate the fragility that results from this, the source of truth on the validity of software version for purposes of remote trust verification must also be technologically trustworthy.

ADAPT addresses the fragility problem and so is positioned to capture the decentralized software market

This is the key insight for the software disruption this paper describes. No system that fails to provide software upgradability without compromising security can survive and fulfill the needs of developers. ADAPT addresses the fragility problem out of the box and so is positioned to capture the decentralized software market in the long term.

10. Data fabric

Data underlying decentralized applications cannot be protected in its entirety with a single set of security guarantees

ADAPT is a framework for building the complete data fabric underlying technologically trustworthy (decentralized) applications. We note that in web3 the data fabric provided by blockchain is incomplete in that it only provides a mechanism for managing globally-synchronized data. Other data components of the web3 software stack are not decentralized: key management components, static data stores, front-end software packages, etc.

Data underlying decentralized applications cannot be protected in its entirety with a single set of security guarantees. Within the same application, some data requires strict privacy, some data requires global synchronization, some data requires remote computation trust, and some data requires a combination of these. ADAPT offers a mechanism to implement all of these possibilities using the same programming approach, in which the logic of the data component is implemented in the ADAPT's programming language, irrespective of the kind of security needed for this component. The framework then allows developers to deploy such components into one of many possible security environments.

11. ADAPT programming

We proceed by describing at a high level ADAPT's programming language and execution environment.

ADAPT is designed as a system with strong emphasis on security. As such, the programming model for the ADAPT containers is critical, as it should strongly support predictable evaluation with highly restrictive features. The design also calls for strong verifiability of the code, for example during the attestation process between ADAPT nodes.

These requirements are not optional good-to-haves, but are critical considerations, without which the system would not possess the strength of the security model fit for the task at hand. Consequently, it would not be useful or practical to adopt one of the existing general purpose scripting language (such as Python or Javascript) to the goal of ADAPT database programming. These languages, having the design goal of generality, provide the opposite of restrictiveness we need, and if they were to be employed in this context, we would have to engage in the continuous game of whack-a-mole to eliminate all the security holes they create, as a matter of on-going practice. This of course will never lead to reliable and provable security, something we strongly aim for.

These considerations have led us to introduce a special purpose programming language into the ADAPT programming environment. The language, called MUFL (pronounced like "muffle"), is a no-side-effects functional programming language. MUFL containers features necessary for the design we describe, including being able to define the container API, use advanced cryptography primitives, interact with other containers, and so on. The language gives developers no access to any IO or system features. Its evaluation result is 100% deterministic and portable across all environments, including the front-end containers.

ADAPT programming environment is as an off-chain equivalent of blockchain smart contracts. As such, using a custom-built programming language is validated by the fact that most blockchain system have opted to develop a special purpose language for their needs, starting with Ethereum's Viper and Solidity, Kadena's Pact, and others.

Figure 8 shows a small snippet of MUFL code that declares a portion of the network API of a database node related to establishing a contact with another database node through a trusted "introducer" node.

```
TRANSACTION register_authorizing_container ($origin -> 2)
TAKES _: ($ad -> ad: address_document::t_address_document) DOES {
    IF m_my_address_document == NIL THEN
        produce_my_address_document().
    END
    // add authorizing container
    add_authorizing_container ad.
    // send request to this container to sign address document
    cid = (ad $identity $container_id) SAFE(hash_code).
    encrypted_trn = transaction::encrypted ($cid -> cid, $trn -> ($name -> "::authorizing_container::sign_ad", $targ -> ($ad -> m_my_a
    RETURN transaction::success [transaction::send cid encrypted_trn].
}

TRANSACTION handshake_init ($origin -> 2)
TAKES _:($target_container_id->cid: hash_code) DOES {
    RETURN handshake::send_ad_and_signatures cid TRUE.
}

TRANSACTION handshake ($origin -> 2)
TAKES arg:($ad->ad: t_address_document, $signature->crypto_signature) DOES {
    handshake::check_ad_signature arg.
    cid = ad $identity $container_id.
    RETURN handshake::send_ad_and_signatures cid NIL.
}

TRANSACTION handshake_final ($origin -> 2)
TAKES arg:($ad->t_address_document, $signature->crypto_signature) DOES {
    handshake::check_ad_signature arg.
    RETURN transaction::success [].
}
}
```

Figure 8: A small snippet of MUFL code.

The following unique features of MUFL serve the overall design goals of ADAPT:

1. MUFL code compiles into a data structure (evaluation tree) that is easily reverse-engineered and becomes human readable, close to the source code. This feature makes it hard to hide exploits when source code is not available.
2. MUFL code forms evaluation units, code packages that can be used to drive individual data containers. Evaluation units are content addressable. The data structure generated by the compiler is a portable Merkle tree, and can consistently generate a hash root that uniquely identifies the evaluation unit. This feature is used by node-to-node attestation, allowing nodes to communicate to each other the exact code running inside their respective databases.
3. Because MUFL code is required to always evaluate deterministically, the programmer does not have access to any system primitives, even system time. The transient data, such as the transaction timestamp and entropy used to create cryptographic keys is generated by the environment into which the container is embedded, and is deterministically accessible inside the database code. This enables the framework to easily create opaque mirrors and encrypted transaction log backups.
4. MUFL is a functional language, but all values in the lambda frames are bound by value. This eliminates the need for a garbage collector, improving overall performance.
5. There is a consistent C++ API that enables the platform to easily add new primitives, such as cryptography functions, and to define new data types. This makes the platform easily extensible.

Summary

ADAPT is a developer framework for building software which reduces the agency of the system's operators by eliminating a monolithic web back end, and replacing it with a data mesh of cryptography-enabled programmable nodes. Inspired by web3, ADAPT, however, recognizes that security is not limited to the double-spend problem solved by blockchain. Other types of security (confidentiality, remote trust) must provided in order to enable important use-cases both in web3 and in enterprise technology.

ADAPT is an alternative form of a decentralized network that decouples applications from each other, while maintaining interoperabilty. Unlike blockchain ecosystems, which add smart contracts on top of a consensus mechanism, ADAPT supports a range of security models built around a universal smart contract layer.

ADAPT is a hybrid system that includes an open source software SDK and includes a set of decentralized services that solve some problems universal in the context of distributed node networks, for example the fragility of remote trust when placed in the context of a healthy software lifecycle.

The vision for ADAPT is that of end-to-end decentralization that includes enterprise and consumer applications alike. It is especially suitable for use cases that are not well-served by blockchain systems, such as data sharing, identity, document sharing, authentication, derivative finance, lending, supply chain, and others.

LIST OF MATERIALS