Privacy Given Strongest Authenticity and Confidentiality

SPAC (Secure Privacy, Authenticity, and Confidentiality)

Version 0.2.4 (Original draft 2023/03/25)

PAC Trilemma

In my opinion, the three properties, authenticity, confidentiality, and privacy inhabit a trade space. I have posited the PAC Theorem or PAC Trilemma to qualify this trade space. The PAC theorem/trilemma states:

> One can have any two of the three (privacy, authenticity, confidentiality) at the highest level but not all three.

The trilemma insists that one must make a trade-off by prioritizing one or two properties over a third.

The ToIP design goals reflect that trade-off and provide an order of importance. The design goals indicate that one should start with high authenticity, then high confidentiality, and then as high as possible privacy, given there is no trade-off with respect to the other two. The properties are defined as follows:

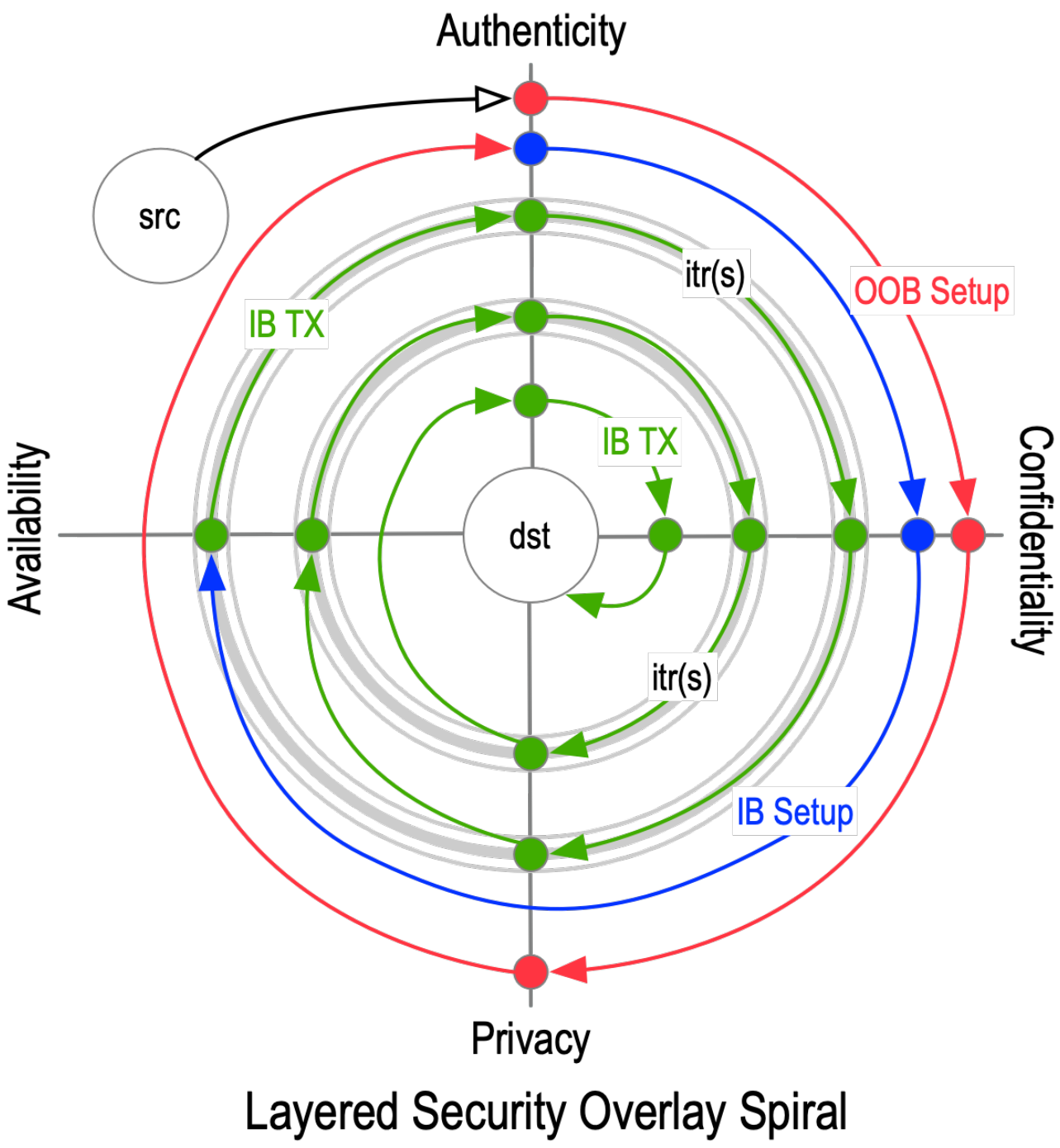
> With regard to the first design goal, establishing trust between parties requires that each party develop confidence in the following properties of their relationship:
>
> 1. Authenticity: is the receiver of a communication able to verify that it originated from the sender and has not been tampered with?
> 2. Confidentiality: are the contents of a communication protected so that only authorized parties have access?
> 3. Privacy: will the expectations of each party with respect to the usage of shared information be honored by the other parties?

> Note that, in some trust relationships, confidentiality and privacy may be optional. Thus our design goal with the ToIP stack is to achieve these three properties in the order listed.

The prioritization of authenticity first, confidentiality second, and privacy third does not mean that when constructing a message that the order of layering of mechanisms in the message that contribute to authenticity, confidentiality, and privacy protection must or should follow that same order.

Indeed, it may be best in some protocols if they don't share the same ordering. To be clear, the ordering of the ToIP design goals w.r.t. authenticity, confidentiality, and privacy may be different from the ordering of layers that employ these three in a given protocol.

This distinction, I believe is a source of much confusion in prior discussions and one of the reasons I provided the overlay spiral diagrams. The purpose of the overlay spiral is to illustrate that multiple layerings of authenticity, confidentiality, and privacy protection mechanisms may appear in different orders both in-band (IB) and out-of-band (OOB) with respect to the setup for and the eventual transmission of a message over a given communication channel or channels.



Layered Security Overlay Spiral

SecurityOverlaySpiral

Diagram: Security Overlay Spiral

From a practical perspective, I use the following elaborations to help refine how I think about the three properties.

- Authentic and Authenticity:
  The origin and content of any statement by a party to a conversation are provable to any other party. Authenticity is primarily about control over the key state needed to prove who said what in the conversation via digital signatures. In other words, it is about secure attribution.
- Confidential and Confidentiality:
  All statements in a conversation are only known by the parties to that conversation.

Confidentiality is primarily about control over the disclosure of *what* (content data) was said in the conversation and to whom it was said (partitioning). Confidentiality is about control over the key state needed to hide content via encryption vis-a-vis the intended parties.

- Private and Privacy:
  The parties to a conversation are only known by the parties to that conversation.
  Privacy is primarily about control over the disclosure of *who* participated in the conversation (non-content meta-data = identifiers). More specifically, privacy is about managing exploitably correlatable identifiers, not merely identifiers of *who* but also any other type of identifier in non-content metadata, including conversation (interaction) identifiers.

Privacy and Confidentiality

There are also two fundamentally different but complementary definitions of privacy that are relevant to protocol design.

The first (ToIP) definition is about the recipient respecting the data privacy concerns of the sender with respect to the content disclosed to the receiver. The protection comes from the recipient protecting the data privacy rights of the sender once the recipient has received that data. This first type of privacy protection may be achieved independently of how private the communications channel was that resulted in the receiver obtaining the disclosure of the content.

The second definition is about the correlatability of the publically viewable metadata included in the communication. This is called non-content meta-data by the surveillance community. This definition distinguishes between confidential data (meta-data or otherwise) that is not publicly viewable, where public means any third party not a party to the conversation. Typically confidentiality protection comes from encryption (i.e., cipher text vs. plain text). But an out-of-band (OOB) exchange of information is confidential (not viewable) by a third-party surveillor of an in-band (IB) channel, thus making the OOB exchange confidential without encryption.

The terms confidential and private also have distinct legal definitions. The legal definition of confidentiality, however, is consistent with the surveillance definition above as well as the definition used by the ToIP properties above. Confidentiality law is about contractual agreements governing the disclosure of data by one party to another. The legal definition of privacy, on the other hand, is not so consistent. The regulatory surveillance law definition is different from the regulatory privacy rights law definition. The former is about drawing the line between non-content metadata and content data in order to be classified as searchable or not (see content-metadata-line and surveillance) and the latter is mainly about protecting personal data rights by data holders, controllers, and processors (see GDPR).

Of interest is the fact that the former (confidentiality law) can be used better to protect the latter (personal data rights). For example, the well-known concept called chain-link confidentiality, circa 2012, details how one can protect the data privacy concerns of the sender via contractual confidentiality law. ACDCs, for example, use the well-known Riccardian Contract formalism, circa 1996, to support chain-link-confidential through what is called contractually protected disclosure.

To summarize:
* Physical Search: (analogy to Constitutionally protected physical search)
Outside of home vs. Inside of home.
The address is a public identifier used for routing and taxing.
Viewing who enters the home from outside via public street is unprotected. (subpoena)
Viewing what happens behind closed doors inside the home is protected. (warrant)

- Information Search:
  Relatively weak legal protection (subpoena) for identifiers, (non-content meta-data)
  because identifiers are needed for public routing and billing.
  Relatively strong legal protection (warrant) for content because the content is not needed for public routing and billing.

But for the purposes of protocol design the definitions of confidentiality and privacy that are most suitable are the surveillance definitions. To clarify, privacy is about the leakage of exploitably correlatable metadata about the parties to the conversation and confidentiality is about restricting the viewability of the content of the conversation to the intended parties to the conversation. In simple terms, privacy is about protecting the knowledge of who is a party to a given conversation and confidentiality is about protecting the knowledge of what was said in that conversation.

The surveillance distinction between privacy and confidentiality has a very practical basis. Recall that for the purpose of protocol design, we defined privacy as protecting knowledge about the parties to a conversation (exchange of data). A primary use case for identifiers in communication protocols is to address the parties to the conversation. Indeed the practical need for identifiers to support addressing often makes the privacy protection of knowledge of identifiers as metadata to conversations particularly (in many cases pathologically) difficult.

Historically, an address of any kind (including the original postal address) exists for two main purposes.

1. So that stuff can get routed (delivered) to that address as a destination via public roads.
2. So that legitimate parties can tax or bill efficiently other parties at any or all addresses.

In the electronic communication world, the same applies. Addresses need to be public so that communications can be routed to destinations over multiple disparately owned, controlled, and regulated communications infrastructures (largely public-regulated monopolies). Likewise, the main mechanism for billing for communication services rendered is address based. So addresses have to be public, not confidential, not only so that stuff can get routed to those addresses but also so they can be the destination for bills to pay for the communication itself. So pragmatically, the legal surveillance world respects that these two purposes (routing and billing) require identifiers (addresses) that must need be public and eaves-droppable.

This is merely a logical, practical extension of property rights w.r.t. unreasonable search. The right to be protected from search only extends to what is not viewable behind closed doors. Anything viewable from a public street is fair game with no protection. Likewise to some extent an automobile or delivery van. What's locked in the trunk or the back of the van (unless it's making noise hearable by a bystander) may be protected. Where it goes on a public street is not protected, and its license plate is a publically viewable address proxy.

As a result, both practically and legally, all routing and billing mechanisms assume to a large degree that addresses are not private. Anyone trying to build a house and then telling the post office, that it can't assign them a public address, or expecting that anyone will build infrastructure to support getting stuff delivered but not using public streets will likely fail. Imagine the cost of supplanting public streets with a totally private conveyance system (air-space, sea-space, and space-space are public already). If one wanted to build a totally private alternative to the public road network then maybe automobiles and vans on private roads wouldn't need license plates (oh wait these are called toll roads, and they need a billing address) so not.

So the analogy holds. We can have strong authenticity and strong confidentiality (of anything but addresses) but weaker privacy w.r.t addresses (identifiers). Beyond a certain point, better privacy over addresses becomes impractical because the world needs addresses for routing and billing on public infrastructure and will always need them. So privacy, as non-correlatable identifiers (metadata) will always be difficult. Whereas, privacy as control over one's data rights is easier because it's not about routing or billing. But we don't have to give up completely on privacy as non-correlatable identifiers if we better qualify it to be privacy as protection against the exploitation of correlatable identifiers (metadata). In simple terms, the goal becomes limiting the exploitation via exploitably correlatable identifiers.

Privacy as Protection Against Exploitation

To better manage this difficult trade-off between the need to address the parties to a conversation and the need to restrict the disclosure of knowledge of those parties to the parties themselves (i.e., privacy) we focus our attention on managing exploitably correlatable non-content metadata, specifically ***exploitably correlatable identifiers***. Thus if we can protect against exploitation derived from the correlation of identifiers we may achieve effective privacy.

Protection against exploitation via correlation provides a more gentle trade space than mere protection against correlation. The former is practical and amenable to cost-benefit analysis, but the latter is largely impractical.

Let's return to our public road delivery analogy. We can achieve effective privacy over public roads if we use a delivery service. The delivery service uses public roads, but the contents of the delivery vans are not viewable by bystanders (third-party observers). Although a bystander can see the van enter and leave our property they can't see the boxes the van delivered or picked up from our property. The van has a public address (its license plate). The boxes the van picks up or delivers all have public addresses. But the addresses on the boxes and the contents of the boxes are not viewable by the bystander. The van makes stops at many houses in the neighborhood which provides herd privacy with respect to correlating van deliveries and pickups to boxes. Analogously the contents of the van may be considered content whereas the van itself is non-content metadata. The van provides a virtual private delivery network over public roads. The closest analogy in the communication world is a Virtual Private Network, (VPN). The limitation is that we have to trust the delivery driver of the van not to tell our neighbors the address information on the boxes delivered or picked up from our address. The driver can't tell what's in the boxes, merely who they came from or who they are being sent to.

Let's extend the delivery driver analogy. Suppose that a malicious adversary replaced the driver with one of its own so that it could view all the addresses of all the boxes inside that delivery van and maybe even open the boxes to view their contents. Now any trust in the delivery van service is broken because of the malicious impersonation of the delivery van driver. We must be careful not to mistake this as a privacy problem but to see

it as an authenticity and confidentiality problem. If the delivery van driver is not impersonatable because any entity at any pickup address can securely authenticate the driver then the adversary will not be able to pick up boxes much less view their addresses or contents. The exploitability of the public addresses on the boxes is derived from the exploitability of the delivery service's authenticity and confidentiality, not the inherent correlatability of the boxes' public addresses.

At first blush, it seems that the solution to protecting address privacy is not a private road network but un-impersonatable delivery van drivers with really secure delivery vans and really secure boxes inside the delivery vans. We would be better off building more secure more trustworthy delivery services (VPNs) that use public roads than building something more complicated and harder to adopt like replacing public roads with private roads (distributed private communication networks). For example, the WireGuard VPN protocol is a best practices VPN from the standpoint of modern pre-quantum security with a path to post-quantum security. It is open source and uses libsodium constructs. We should not reinvent that wheel.

Cold War versus Hot War

In a strong sense, given the definitions above, authenticity and confidentiality protection are in a different class or regime from privacy protection.

Authenticity and confidentiality are like a ***cold*** war. Authenticity overlays based on public key digital signatures provide arbitrarily strong protection with minimal resources. Likewise, confidentiality overlays based on public key encryption provide arbitrarily strong protection with minimal resources. To reiterate, the reason authenticity and confidentiality are analogous to a cold war is that we have existing tooling that enables cost-effective cryptographic strength authenticity (as in end-verifiable attributability to an AID) via digital signatures and cost-effective cryptographic strength confidentiality (as in secure end only viewability by an AID) via asymmetric en-de-cryption. This strength is such that attack is computationally infeasible against the crypto and the only weakness is in the strength of the key management of the associated key pairs. So there need be no casualties in this cold war. The only looming vulnerability is that a secret weapon might be developed (aka a quantum computer) which would break the existing crypto. But given that we now have quantum-safe, signatures and asymmetric encryption, as long as the key management system is sufficiently agile and post-quantum proof (like KERI), it will remain a cold war because one just switches to the quantum-safe crypto once someone has built a quantum computer capable of breaking the existing crypto.

In comparison, privacy is like a ***hot*** war. Rapidly evolving tactics and technology make privacy protection like a resource-constrained war of attrition against vastly superior opponents. We have to be careful because decreasing the privacy attack surface may actually increase the net total harm due to lost value capture opportunities. To reiterate, the reason privacy (as exploitable correlatability of identifiers) is a hot war is that casualties are unavoidable. The strategy, tactics, techniques, resources, and incentives to expend those resources for correlation are a function of the particular adversary. So the protection tactics and strategy are specific to the type of adversary. It becomes more about minimizing the casualties and the expended resources so as not to lose the war.

Suppose we want to build a communications channel that is not exploitable via correlation of the identifiers. Because the communications must be routed between end-points, all of the routing identifiers, the routing tables, and every intermediary are all attack surfaces for correlation. The receiver and sender are each

themselves an attack surface for correlation independent of the channel.

The channel defines the boundary between in-band and out-of-band. For some adversaries even out-of-band mechanisms are correlatable. So if our protection must account for any and all correlation mechanisms from any and all types of adversaries including those with virtually unlimited resources to correlate both in-band and out-of-band then there is no practical solution to the privacy protection problem. One can always play the equivalent of a trump card that says, here is a way some adversary given enough resources can correlate, and so the protocol is broken in terms of privacy. This is why the qualifier *exploitable correlatability* is used. The type and degree of exploitability depend on the adversary.

As a result, we must make risk mitigation trade-offs between exploitable harm and protection cost by focusing on the exploitable correlatability of identifiers used to convey information over a communications channel.

Let's use a related war analogy, suppose one is designing a tank and needs to select the thickness of the tank's armor. One can't determine the armor thickness needed until and unless they scope the types of attacks that the tank must be able to withstand. If one of the allowed attacks is a nuclear bomb, then the tank designer has an impossible trade-off because a tank by definition is a type of mobile artillery. Therefore, any armor thick enough to protect against a nuclear bomb would be too heavy to move.

In the hot war of privacy, we have to Pareto optimize protection mechanisms. Notwithstanding the limited resources at our disposal, we can still protect against many types of attacks from many types of adversaries but not all attacks from all adversaries.

One way to approach this is to use a risk mitigation rating and ranking process. In critical systems, one standard process rates and ranks risks by quantifying the magnitude of the harm for a given risk and then multiplies it by the likelihood of the harm. Risks with high harm and high likelihood are ranked ahead of risks with high harm but a lower likelihood.

A cost-benefit analysis is then performed on the relative cost of any mitigation measures as a percentage of all available resources to mitigate all harms from all risks. A comet striking the earth, for example, might have very very high harm but its likelihood is so small and the cost of mitigation so high that it's not a risk worth mitigating.

In any resource-constrained decision problem, we have to make trade-offs, and we need to understand the trade space well if we want to make optimized trade-offs. In a later section, we will discuss threats and mitigations to better understand that trade space.

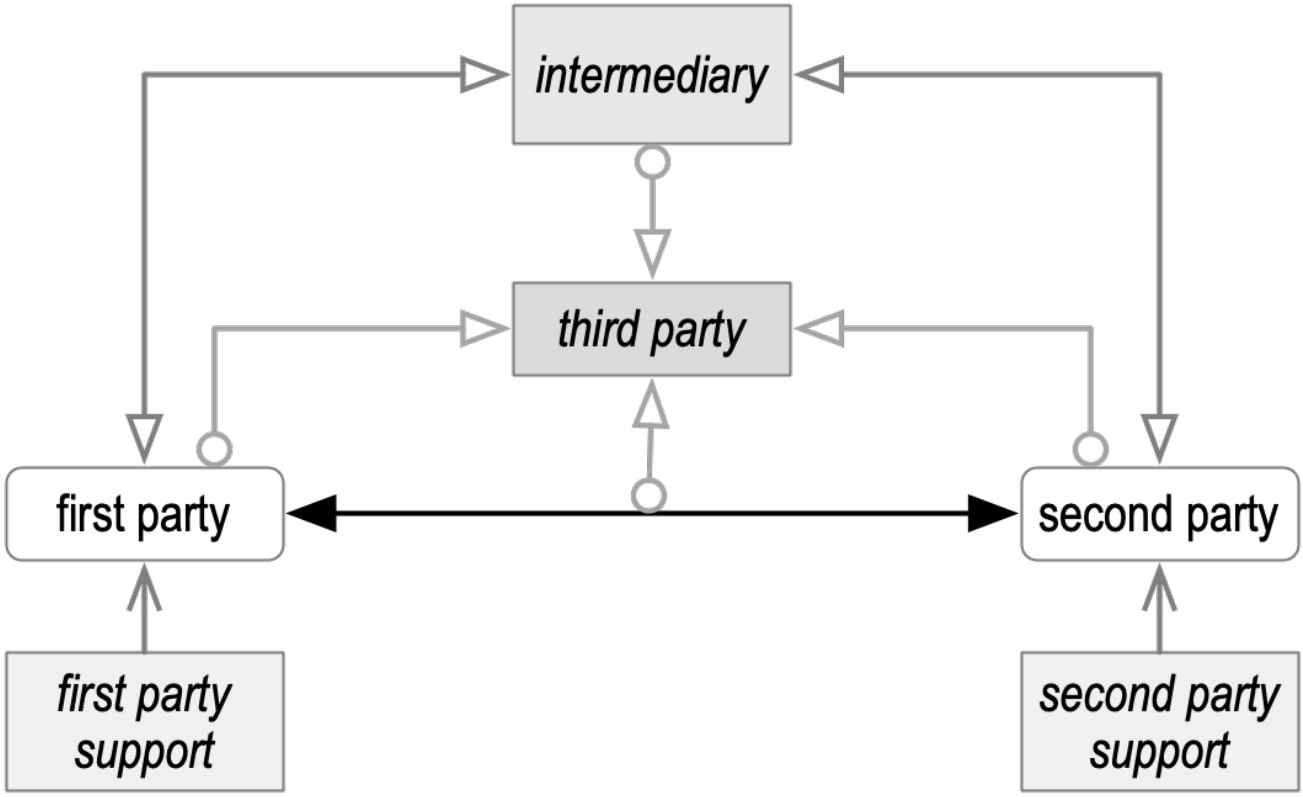
On the other hand, in the cold war of authenticity and confidentiality, even with limited resources, we can protect against almost all attacks from all adversaries. In the next section, we will detail specifically what the strongest practical authenticity and confidentiality protection looks like.

Three-Party Exploitation Model

For this analysis, we use the three-party disclosure and correlation exploitation model. The 1st and 2nd parties represent the parties to a conversation or exchange. The 1st party discloses data to a 2nd party. The 1st and 2nd parties have knowledge of each other via their identifiers. The 1st party as discloser has knowledge of the

disclosed data (content). As a result of the disclosure the 2nd party as disclosee also has knowledge of the disclosed data (content). The identifiers are non-content meta-data, and the disclosed data is content data. A 3rd party is any party that is not either a 1st or 2nd party. An intermediary may be considered a 1st, 2nd, or 3rd party depending on how and by whom it is trusted or not.

Privacy with respect to the 3rd party is protected if the 3rd party has no knowledge of the identifiers used by the 1st and 2nd parties for the conversation (disclosure). Confidentiality with respect to the 3rd party is protected if the 3rd party has no knowledge of the disclosed data (the content data disclosed). A 3rd party may directly break privacy by observing messages that contain the identifiers of the 1st and 2nd parties. A 3rd party may directly break confidentiality by observing the content of messages between the 1st and 2nd parties. In addition, the 3rd party may indirectly break both privacy and confidentiality by collusion with the 1st or 2nd party or via collusion with an intermediary. This is diagrammed below.



ExploitationModel

Diagram: Three Party Exploitation Model

Exploiters

The three main classes of potential exploiters are as follows:

1. State Actors (3rd party):
   The primary incentives for state actors are political, but some like North Korea are also monetary. North Korea is responsible for a significant percentage of ransomware attacks (see NK Ransomware, NK Success).
   State actors may use any combination of attacks to surveil, regulate, and persecute all first parties,

second parties, and intermediaries with virtually unlimited resources and the ability to cause harm. Because of the virtually unlimited resources and techniques that state actors can apply we will treat them as out of scope for mitigation for the time being. That does not mean we will not consider state-actor resistant mitigations but to attempt state-actor proof mitigations would likely take us down rabbit holes that we can never return from.

2. Identity Thieves (3rd party):
   The primary incentive for identity thieves is monetization through asset theft. Typically either directly through the capture of access credentials to assets or indirectly through the sale of personally identifying information (PII) that may be used to capture access credentials. Another primary monetization strategy is ransomware of either access credentials or PII that may be used to capture access credentials or simply sensitive PII such as health data.
3. Aggregators (3rd party, 2nd party, 1st party):
   The primary incentive for aggregators is to sell data to advertisers. This means profiling groups of potential customers in attractive demographics. This profiling may be entirely statistical. There are several classes of aggregators. These include:
   - Intermediaries (3rd party):
     - Search as an aggregator
     - Cloud provider as an aggregator
     - ISP as an aggregator
   - Platforms (2nd party)
     - Colluding 2nd parties as aggregators
   - Inadvertent (1st party)
     - Inadvertent colluding 1st party as an aggregator via web cookies

Identity Theft: Threats and Mitigations

Threats

For this analysis, we assume that content data is confidentially encrypted between
1st parties and 2nd parties when sent over the wire. The identity thief cannot view this data without first compromising 1st party or 2nd party credentials in order to either decrypt the data or view it in its unencrypted state on a storage device. The ultimate target is a treasure trove of sensitive data held by a 2nd party. This may be health or other data that may be captured as ransomware, or it may be credentialing information that enables first-party impersonation in order to either directly capture 1st party assets or may be sold to secondary thieves who then use it to capture 1st party assets. Another target is the asset access credentials of high-value 1st parties. Notable is that identity thieves do not monetize aggregated content data or metadata (anonymized or not) by selling it to advertisers. The return on investment for data aggregation is too low for identity thieves in general.

The primary attack mechanism is labeled differently in different venues but may be described as a multi-step recursive privilege elevation attack (authentication and authorization). The attacker compromises weak access credentials on low-value targets that allow it to elevate its privileges in order to compromise a higher-value target until it finally has the access credentials of a treasure trove. An example of this would be to compromise

the VPN login of some customer or employee of some company that allows an attacker to compromise some other employee's VPN login to a connected company and so on until it captures access credentials to a connected company's treasure trove. It starts with what is called an edge attack on the weakest company. The elevation mechanism often leverages what is called a BOLA or BUA vulnerability in the access API of a given company. These vulnerabilities singularly or in combination allow an adversary to elevate their access privileges (see BOLA Explained. The OWASP project lists in order the primary API vulnerabilities that enable a recursive privilege elevation attack. These are not privacy weaknesses but authentication-based authorization weaknesses.

This is described thusly:

> Hackers can use data stolen from companies with weak security to target employees and systems at other companies, including those with strong security protocols. … the data ecosystem has become so vast and interconnected that people are only as safe as the least secure company that interacts with any company that has access to their data. That "least secure company" does not even need to have access to the consumer's data. … There were 5,212 confirmed breaches in 2021, which exposed 1.1 billion personal records across the globe" (see data security)

To summarize, a given trust domain is only as strong as the weakest connected trust domain (in which connected is recursively applied).

A recursive privilege elevation attack usually starts with a 2nd party impersonation using phishing or smishing to steal 1st party credentials by spoofing an impersonated 2nd party. Now that the attacker has 1st party access they can leverage that to discover other 1st parties that have access to higher value connected clients, then rinse and repeat.

A related elevation attack is to attack a weak intermediary which may be used to stage a more sophisticated 2nd party impersonation attack. The classic DNS hijack is such an example. An adversary hijacks (using cache poisoning or some other mechanism) the DNS server of some 2nd party. The adversary then requests a new DNS certificate from a CA. The CA verifies control of that DNS domain by querying the DNS Zone file for the impersonated 2nd party secret provided for authentication. The hijacked DNS returns the secret thereby fooling the CA into issuing a valid certificate to the adversary. The adversary then hijacks the DNS server of a first party so that when they go to login to the 2nd party host they are redirected to a host under the control of the adversary. The 1st party verifies the valid DNS credential miss-issued to the adversary and then logs in thereby allowing the adversary to steal the 1st party credentials. The adversary then logs into the 2nd party host with 1st party access. This can be repeated.
A related attack is to first compromise an intermediary using either a direct or combined attack and then leverage the surveillance of first-party to second-party traffic via the now compromised intermediary to mount attacks on the second-party in order to elevate to first-party credentials.
A more sophisticated attack is a code supply chain (intermediary) attack that creates a back door to capture first-party credentials.

Mitigations

- Use strong authentication by 2nd parties of any first party at any step along a multi-step recursive privilege elevation attack. In any multistep recursive privilege elevation attack any step that has strong

authentication with individually signed requests for authorization will stop the attack from proceeding.
- Use strong authentication by any 2nd party of any 1st party for any exploitable use of first-party credentials, whether those credentials be credit card, bank account, or access to sensitive data.
- Use strong authentication (zero-trust data in motion and at rest) at any party holding a treasure trove. Strong authentication is also state-actor resistant.
- Get rid of treasure troves of access credentials by using decentralized trust domains (DPKI). This means passwordless security with strong key rotation mechanisms. This is also state actor resistant.

Aggregation: Threats and Mitigations

For this analysis, we assume that content data is confidentially encrypted between
1st parties and 2nd parties when sent over the wire. The aggregator must either merely aggregate the correlatable information contained in the non-content meta-data such as identifiers or must be a 2nd or 1st party with access to 1st party data. The aggregator needs voluminous data in order to provide sufficient monetization via advertising. This means attacks to compromise 1st or 2nd party credentials are too costly given the amount of data captured by such attacks. Moreover, advertisers generally are legitimate companies that would incur a huge legal liability should they engage in direct attacks on 1st party credentials. Their primary mode of operation is to incentivize first parties to disclose exploitable information either simply as correlatable metadata or as content data. Typically, data may be anonymized and then re-correlated to demographic pool sizes appropriate for advertising campaigns. As mentioned above there are several classes of aggregators each with a different exploit mechanism.

- Intermediaries as Aggregators
    i. Search as an intermediary (3rd party).
    In this case, an internet search engine is able to track the click-throughs of a successful search result. This enables the search engine to monetize the 1st party searcher by correlating the meta-data of the incoming client browser with the 2nd party endpoint resulting from the search.
        - Mechanisms:
            - 1st party permissioned surveillance of first party metadata collected by 3rd party search intermediary and sold to other 3rd parties
        - Mitigations:
            - Use VPN in combination with a partitioned 1st party identifier such as a random email address
        - Use a search engine that contractually protects 1st party search metadata.
    ii. Cloud data storage provider as an intermediary (3rd party).
    In this case, all 1st party content data stored in the cloud may be anonymized and re-correlated to provide monetizable advertising campaign pools.
        - Mechanisms:
            - First-party permissioned surveillance of first-party content data by 3rd party cloud provider intermediary and sold to other 3rd parties.
        - Mitigations:
            - Use cloud storage providers that use end-to-end encryption (first-party confidential content data)
            - Use cloud storage providers that contractually protect 1st party content data.

iii. Internet Service Provider (ISP) as an intermediary (3rd party).
In this case, ISP has access to all traffic routed through it for 1st party client connections to any 2nd party. ISP can anonymize and then re-correlate into monetizable advertising campaign pools.
    - Mechanisms:
        - Non-content Metadata (routing and billing) identifiers. In the USA for example any anonymized data (both metadata & content) is aggregation permissible for any ISP.
        - Mobile device location tracking.
    - Mitigations:
        - Use an ISP that contractually protects user metadata
        - Use VPN that contractually protects user metadata. A VPN provides herd privacy relative to ISP tracking of the source to 2nd party destinations.
        - There is no mitigation for mobile device tracking by mobile service providers except to turn off the mobile device.
        - Use distributed decentralized confidential network overlay to make metadata harder to correlate. This could include onion or mix network routing such as TOR, Elixxir, MainFrame, and DIDComm Routing. In general, a decentralized network overlay is difficult to incentivize and as a result, may have relatively low adoptability when compared to a VPN.
        - Use information-theoretic secure routing. An example would be random walk routing. Of all the mitigations, this one is state actor resistant. Information-theoretic secure routing, however, has similar adoptability problems as a decentralized confidential network overlay.

iv. Platform as an aggregator (2nd party).
In this case, the platform is a 2nd party with access to all the 1st party data disclosed to it. This includes both non-content metadata and content data. The platform anonymizes the data and re-correlates it into monetizable pools for advertising on the platform itself. A related approach is for smaller 2nd parties to collect their data and sell it to 3rd party data aggregators, who then package it for 3rd party advertisers. Statistical techniques like regression enable monetizable aggregation even with weakly correlatable data given enough 2nd party data. (see 1st party data correlation), 0th party data
    - Mechanisms:
        - First-party permissioned content data and non-content metadata are anonymized and then statistically regressed to form monetizable advertising campaign pools. Even anonymized 1st party data without correlatable identifiers have sufficient exploitable correlatabilty for monetization. Privacy-protected metadata (identifiers) are insufficient to protect against exploitation. Data anonymization is not protective (fallacy of K-Anonymity).
    - Mitigations:
        - Don't interact with 2nd parties that aggregate without permission.
        - Interact only using contractually protected disclosure alone or in combination with contingent or selective disclosure.

v. Inadvertent 1st party as aggregator (1st party, 2nd party, 3rd party).
In this case, the 1st party uses web browser cookies or other permissioned mechanisms that make the 1st party itself the tracker that may be exploited by either 2nd parties or 3rd parties that may glean data from the tracking mechanisms. The data is typically metadata but may also be content

data depending on the invasiveness of the cookies.

- Mechanisms:
  - Browser cookies make first-party browsers an inadvertent permissioned intermediary that can leak both non-content metadata and content data to 2nd parties and/or 3rd party intermediaries. This enables those parties to aggregate 1st party data and monetize it for advertising campaigns.
- Mitigations:
  - Don't use browsers in a way that supports tracking first-party data via cookies by 2nd parties or 3rd party intermediaries. All the major browsers now support a type of incognito mode that prevents the use of cookies for tracking. Most 2nd party websites support disabling most cookie-based tracking.
  - Use browsers that contractually protect 1st party data and anonymize the 1st party source identifiers.
  - Use VPNs that contractually protect 1st party data and anonymize the 1st party source identifiers.

Summary Threats and Mitigations

As the analysis above shows the most fruitful application of TSP-based protocols with regards to exploitably correlatable identifiers may be in improving VPNs to limit aggregation for advertising by 3rd parties or in providing TSP-based protocols that enable VPN like trusted intermediaries for herd privacy against 3rd party aggregators. Whereas protection from 2nd party aggregators may be best provided by higher layer protocols that provide contractually protected disclosure of 1st party data to 2nd parties. Finally, protection from identity theft may be best provided with TSP-based protocols that provide strong authenticity and confidentiality independent of correlatable identifiers. Proper full zero-trust (verify everything) approaches with cryptographic roots-of-trust for authentication, and authorization simplifies API design and may mitigate the most common vulnerabilities (including BOLA and BUA).

Unbounded-Term AIDs instead of Long-Term Public Keys

KERI AIDs (autonomic identifiers) have a novel and unique feature called persistent controllability. With KERI the controlling key pairs of an AID may change via rotation or in other words the control is transferable to a new set of controlling key pairs. This enables the controller to maintain persistent control over the AID in spite of the compromise of the controlling key pairs. Specifically, the KERI pre-rotation mechanism uses one-time pre-rotated rotation keys to recover control of the AID by transferring that control to a new un-compromised set of key pairs. Because the pre-rotated keys are not exposed they are much harder to attack. At inception, a persistable (transferable) AID makes a forward commitment to the rotation key pairs via cryptographic digests of the public keys. These keys are never used until and unless there is a rotation. Only the pre-rotated keys will verify against their digests. With each rotation, a forward commitment is made to a new set of one-time-only pre-rotated keys thereby always staying one step ahead of an attacker.

KERI's pre-rotation may be characterized as what many in the academic literature now label a meta-cryptographic system. KERI itself only uses well-established cryptographic operations, like digests and digital signatures but uses them within a system that provides useful features like transferable or persistent control over an identifier.

Many public key cryptographic algorithms employ the concept of *long-term* public keys and *short-term* public keys. Often long-term keys have less exposure than short-term keys so they can maintain their cryptographic strength longer. Typically this is to better manage the friction of key exchange where higher friction authentication of long-term keys is used to enable or bootstrap cryptographic operations using short-term or ephemeral keys.

A long-term public key is often used as the primary identifier for a party in an interaction or exchange. With this usage the long-term public key may be called the address of the party. The problematic limitation of using long-term public keys as party identifiers (addresses) is that the public key is ***only*** long-term not unbounded-term. Eventually, any long-term public key must be abandoned due to exposure. NIST recommends that public authentication keys have a cryptoperiod (time before key must be replaced) of no more than 1 to 2 years. This means that any authentication or reputation associated with a public key (address) by a relying party (a public key as an identifier for a given party) must also be abandoned. This forces the two parties to start over by re-authenticating or re-registering a new public key as a replacement address or identifier. Moreover, there is no inherent way for a relying party to know that some other party has abandoned its public key as an identifier. This exposes the two parties to attacks that exploit this uncertainty via the use of stale public keys and/or attacks on the re-registration process.

This is one of the reasons why we consider key rotation the hard problem of key management. KERI solves the hard problem of key rotation using key pre-rotation in combination with a cryptographically verifiable append-only data structure based on a cryptographic root of trust that provides proof of key state for any persistable (transferable) AID. This data structure is called a Key Event Log (KEL). A controller publishes its current key state via its KEL, thereby enabling any relying party to verify the current state and also to detect when a key has been rotated (abandoned). Stale usage of compromised keys by adversaries is thereby detectable. Pre-rotation protects the re-registration process. Identifiers (addresses) therefore maintain their reputation in spite of key rotations. Indeed transactions can be securely bound to the key state by anchoring them in the KEL.

Another significant advantage of using unbounded-term AIDs instead of long-term public keys is post-quantum safe cryptographic agility. KERI's pre-rotation mechanism is post-quantum safe because cryptographic strength digests such as Blake3, Blake2b, and Sha3 are also strong against quantum attack Hash Collisions. This means that pre-rotated keys are protected from quantum attack exposure and inversion by their digests. This provides sufficient cryptographic agility to be able to rotate to post-quantum safe digital signatures and public keys. This makes the pre-rotated keys safe from surprise quantum attacks. In contrast, long-term public keys as identifiers are vulnerable to surprise quantum attacks.

With KERI, AIDs as primary identifiers may be valid for an unbounded-term instead of merely long-term. This changes the way meta-cryptographic operations that provide authenticity and confidentiality may be expressed. In the following development, in many cases, where a long-term public key would normally be used, we may use an AID with unbounded-term instead. Given the AID, its latest public key may be looked up from its

KEL. This applies to both signing public keys and encrypting public keys. Moreover, with certain caveats, the encrypting public key may be cryptographically derived from the signing public key so only the signing public key needs to be managed in the KEL.

Using an unbounded-term AID with a KEL solves the problem of the un-detectability of abandoned long-term public keys. If the provided message is using stale keys for a given AID the associated cryptographic operations will fail against the current key state, and the relying party can drop the message. Likewise, when the controller updates its key state and uses new keys in a message, but the relying party has a stale copy of the KEL, the associated cryptographic operations will also fail, which indicates to the relying party the need to update its copy of the key state. The controller can also send a hint of its current key state along with any message.

One simplification that results from this approach is that many protocols can be simplified to remove repeated setups using ephemeral (short-term) keys whose purpose is to reduce the exposure of the long-term keys.

End-Verifiabilty and End-Only-Viewability

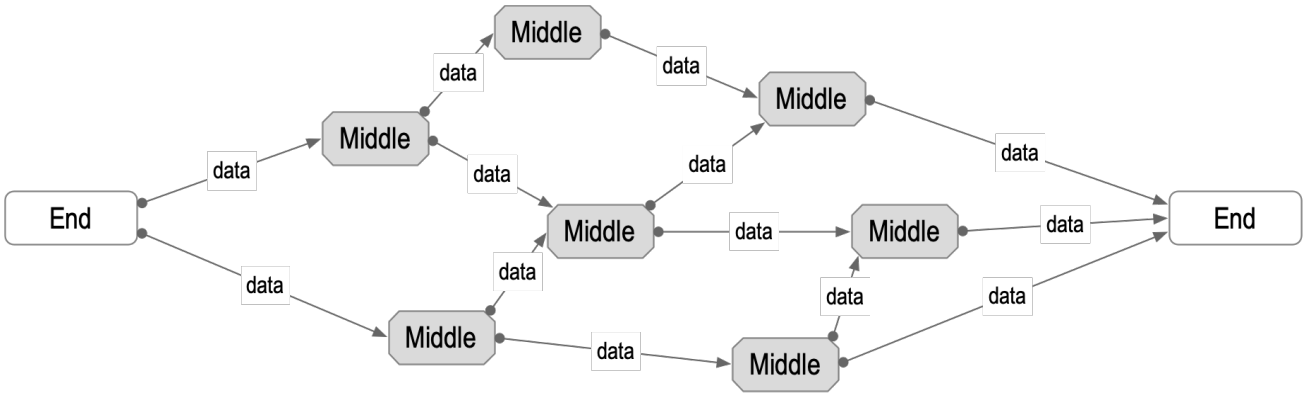
One of the key principles to building secure systems is to minimize the number of different attack vulnerabilities and for those that remain make them as invulnerable as is practically possible. The approach that KERI takes is to minimize the attack vulnerabilities to key management and nothing else and then to make that key management as strong as possible. The principle makes it so that in any exchange of data (communication):

- If the edges (ends) are secure, then the security of the middle doesn't matter.

The corollary to that is that:

- It's much easier to protect one's private keys than to protect all internet infrastructure.

This is shown in the following diagram:



End2EndNetwork

Diagram: End-to-End Network

For authenticity, the way to remove vulnerabilities in the middle is called end-to-end verifiability of authenticity or end-verifiability for short. This naturally leads to what we call ***ambient verifiability*** where any-data can be verified any-where, any-time by any-body.

There is a dual property for confidentiality.
For confidentiality, the way to remove vulnerabilities in the middle is called only-at-end viewability for confidentiality or end-only viewability for short. This naturally leads to the concept that anyone's confidential data is only viewable one-where, one-time by one-other-body.

Together these properties mean we only have to secure key management at the ends (edges) not everywhere in between. Insisting on these principles greatly simplifies and strengthens any protocols we may choose to build.

To reiterate, a confidentiality overlay is the dual of an authenticity overlay.
An end-verifiable authenticity overlay is based on one or more asymmetric key pairs for signing, where only the end with the private key can sign, and any end with the public key can verify. To clarify, only the key pair controller can sign with the private key, any recipient can verify with the public key.
An end-only-viewable confidentiality overlay is based on one asymmetric key pair for en-de-cryption, where any end with the public key can encrypt, and only the end with the private key can decrypt. To clarify, only the key pair controller can decrypt with the private key, any sender can encrypt with the public key.
Any identifier can have a key state that includes both an asymmetric signing key pair and an asymmetric decryption key pair.
Either the key pairs can be the same, or the decryption key pair can be derived from the signing key pair
Thus an efficient approach is one where only one key state, the signing key state, needs to be maintained (with caveats).
Given the identifier and the security overlay's mapping to look up key state, any other party, can either verify with the signing public key that a message was non-repudiably sourced by the controller of the identifier (authenticity) (non-repudiable any-end-verifiable) or can encrypt a message using the encryption public key to ensure that only the controller of the identifier can view it (confidentiality) (restricted only-end-viewable)

To analyze confidentiality we use a 3-party model where the 1st party is the sender, the 2nd party is the intended receiver, and the 3rd party is any unintended receiver.

A system with strong end-only viewable confidentiality may have the following properties:

- 3rd party non-viewability
- 1st party non-viewability (this protects 1st party from liability wrt to 2nd party and protects 2nd party from exploit by 1st party) (see appendix for a more detailed description of this property)
- 2nd party partition-ability by 1st party
- Detectable leakage (2nd party to 3rd party)
- Detectable collusion of 2nd party against 1st party (2nd party with 2nd party, or 2nd party with 3rd party)
- Detectable collusion of 1st party against 2nd party (1st party with another 1st party relative to same 2nd party or 1st party with 3rd party)

Adoptability via Dumb Crypto

There is an explosion of activity toward developing new cryptographic algorithms. Elliptic curve cryptography has proven to be both highly secure and highly performant compared to other approaches. Unfortunately, new cryptographic algorithms take time to season, as it were, that is, to be battle-tested for weaknesses and

exploits. Likewise, it takes time for new cryptographic algorithms to be supported widely across programming languages and operating systems. It does not help much, practically speaking, to design a protocol that needs advanced crypto that is not yet readily available or sufficiently adoptable for some time. It becomes a theoretical protocol, not a very usable one. Hence, a design criterion we employ is that only highly adoptable, well-proven, cryptographic operations from broadly supported libraries be used. If a required cryptographic operation is not in those libraries then it's not a viable candidate for the protocol.

The best example, of best practices in highly adoptable crypto algorithms, is NaCL. The most popular implementation of NaCL is the libsodium library which is supported almost universally across programming languages and operating systems. This does not mean that other equivalently adoptable libraries are ruled out but merely that we use libsodium as a starting baseline or departure point. Furthermore, the signing and encryption algorithms in NaCL (Ed25519 and X25519) have been accepted into NIST's roadmap for approved government use (see Curve25519).

In this work, we only consider pre-quantum safe algorithms. We assume the encoding of any cryptographic primitive supports sufficient agility. If we can design a protocol that only needs the operations found in libsodium then we have a practically adoptable baseline for that protocol.

Best Authenticity (Public Key Signing)

The standard for strong public key signing algorithms is SUF-CMA (Strong UnForgeability under Chosen Message Attack). When a digital signature has SUF-CMA with at least 128 bits of cryptographic strength then it is computationally infeasible to forge a signature or to discover the private key by choosing messages. The Libsodium implementation of the NaCL Ed25519 signature scheme has SUF-CMA Provable Security of Ed25519. So this is our baseline. In comparison, the popular ECDSA signature scheme only has the weaker EUF-CMA (Existential UnForgeability under Chosen Message Attack). Any scheme with SUF-SMA also has EUF-CMA This relative weakness of ECDSA has already been exploited in bitcoin. Consequently, some additional care must be taken when using signature schemes, like ECDSA, that are not SUF-CMA.

The Ed25519 in libsodium has some additional security properties that make it a good starting baseline. The following table lists those properties.

scheme	EUF-CMA	SUF-CMA	S-UEO	M-S-UEO	MBS
Ed25519-LibS	X	X	X	X	X

In addition to SUF-CMA, the S-UEO (Strong Universal Exclusive Ownership) and M-S-UEO (Malicious Strong Universal Exclusive Ownership) properties provide additional resiliency against key substitution attacks. Whereas the MBS (Message Bound Security) property ensures a given signature verifies a unique message even for malicious keys.

In addition, Ed25519 avoids private key memory leakage attacks, and its key clamping mechanism makes the re-use of a Ed25519 key pair for its bi-rationally equivalent X25519 encryption key pair resistant to key leakage attacks via the X25519 key pair.

In summary, the baseline for strong authenticity is to use the libsodium Ed25519 signature scheme

Best Confidentiality (Public Key Encryption)

The standard for strong public key encryption (PKE) algorithm is IND-CCA2 (Indistinguishability under Adaptive Chosen Ciphertext Attack). A scheme that has IND-CCA2 also has the weaker IND-CCA (Indistinguishability under Chosen Ciphertext Attack) which also has the weaker still IND-CPA (Indistinguishability under Chosen Plaintext Attack). What IND-CCA2 means is that an adversary can't submit ciphertexts in such a way as to discover the private key used to decipher those texts even when the adversary has access to an oracle that tells the adversary whether or not the ciphertext was a valid ciphertext. In addition, an adversary can't construct a valid ciphertext unless the adversary is aware of the associated plaintext.

For a more in depth description of IND-CCA2 see IND-CCA2 part 1, and IND-CCA2 part 2

Earlier forms of public key encryption (PKE) only had IND-CPA or IND-CCA. Getting IND-CCA2 requires what is called an Integrated Encryption System (IES) that provides a public key encryption scheme. An IES that uses an elliptic curve is called an ECIES. A notable example of an ECIES that provides a public key encryption scheme with IND-CCA2 is the NaCL/Libsodium *sealed box*. What is special about an ECIES like NaCL's *sealed box* is that it provides the performance of symmetric encryption but with the end-only viewability of asymmetric public key encryption. What we mean by end-only viewability is that only the recipient to whom the box is sealed can decrypt the ciphertext. The sender cannot. The way *sealed box* does this is that it creates a non-interactive Diffie-Hellman (DH) en-de-cryption key using an ephemeral X25519 key pair with the long-term X25519 public key of the recipient. The sender never sees the ephemeral X25519 private key so it can't reconstruct the DH en-de-decryption key. The *sealed box* includes the ephemeral X25519 public key so that the recipient can reconstruct the DH en-de-cryption key using its own long-term X25519 private key and the provided ephemeral X25519 public key. The ciphertext is encrypted using symmetric encryption so it's high performant and only the holder of the long-term X25519 private key for the long-term X25519 public used in the *sealed box* to encrypt, can decrypt the ciphertext.

The following references might be helpful. ECIES properties, ECIES, IES, IES More,
ECIES is IND-CCA2, Crypto Box Seal is ECIES

More recently the more descriptive term HPKE (Hybrid Public Key Encryption) has been used to describe IES that support public key encryption (PKE). This is memorialized in the IETF RFC–9180. RFC–9180 extends basic HPKE with 4 modes of operation in a standardized approach. The Libsodium sealed box has equivalent properties to RFC–9180 base mode [HPKE Analysis](
https://eprint.iacr.org/2020/243.pdf). For a good introduction to HPKE see HPKE and HPKE Blog

Unfortunately, implementations of RFC–9180 are not yet broadly available. So our baseline for IND-CCA2 is the Libsodium Crypto Box Seal or Sealed Box. This uses the X25519 key pairs and the venerable high-performant XSalsa20 stream cipher with the Poly1305 MAC that together provide the XSalsa20-Poly1305 AEAD that underlies the sealed box HPKE.

Furthermore, IND-CCA2, by itself, is insufficient to provide strong authenticity of the ciphertext. As section 9.1.1 of RFC–9180 points out, all variants are vulnerable to key-compromise impersonation attacks that may arise from a compromise of the recipient's private key (see KCI). RFC–9180 suggests that applications that require resistance against key-compromise impersonation should take extra steps to prevent this attack such as using a digital signature generated with the sender's private key over the HPKE container, which means signing the *sealed box* (when using Libsodium). Essentially a KCI attack enables a compromise of the recipient's keys to impersonate the recipient and perform a type of MITM (man-in-the-middle) attack when performing a key exchange where they substitute a different keypair from that intended by the sender. As a result, the sender may not detect the impersonation and therefore will encrypt to an impersonated key-pair see (KCI explained, Pandora's Box). This also allows the attacker to purport that the confidential information was meant for the impersonated key pair thus also breaking authenticity.

Best Combined Authenticity and Confidentiality

As developed above for combined strong authenticity and confidentiality we need both hybrid public key encryption to the receiver's key pair (with IND-CCA2) and public key signing from the sender's key pair (with SUF-CMA). There are several ways to combine a cipher text and a signature. In order to protect against key-compromise impersonation attacks we must sign the encryption with the recipient's public key in plaintext. This is called encrypt-then-sign. But merely signing the encryption does not provide complete protection against all attacks. An attacker can strip the signature and resign the ciphertext with the attacker's key but has never seen the plaintext of the cipher. In some circumstances, this might lead the recipient to falsely believe that the sender had knowledge of the plain text when it did not. For that, we must not only bind the recipient's public key to the signature in plaintext but also bind the sender's public key inside the ciphertext.

Modified ESSR

An approach that protects against both KCI and sender impersonation of the ciphertext is called ESSR for Encrypt Sender's key and then Sign Receiver's key. The ESSR approach is detailed here ESSR. For a more accessible explanation see the following PKAE1, PKAE2, PKAE3

The properties that protect against key compromise impersonation attacks are called TUF-PTXT (Third-party UnForgeability of PlainText), TUF-CTXT (Third-party UnForgeability of CipherText), RUF-PTXT (Receiver UnForgeability of PlainText), RUF-CTXT (Receiver UnForgeability of CipherText). Simply using the encrypt then sign approach provides TUF-PTXT, TUF-CTXT, and RUF-CTXT but not RUF-PTXT.

Signing the receiver's public key in plaintext outside the ciphertext portion of the message binds the receiver's public key making a key substitution attack on the underlying plaintext to the signed ciphertext detectable which gives RUF-PTXT. One weakness of encrypt then sign in the case of a malicious receiver is that if the receiver can substitute a new key pair it can encrypt any message using the sender's public key to the new keypair and purport that the sender intended to encrypt to that new key pair. Signing the ciphertext still leaves a malicious receiver free to find a combination of plaintext and receiver key pair that produces the same ciphertext. This is hard but not computationally infeasible. So we need a way to limit how malicious ciphertexts may be generated, and the ESSR way to do that is to bind the intended receiver's public encryption key to the sender's signature. This way the ciphertext must decrypt using the intended receiver's key pair, or it's invalid.

The security of an encryption system ensures that it is computationally infeasible that given the same key two different plaintexts do not encrypt to or decrypt from the same ciphertext, i.e., a given plaintext must roundtrip through the cryptosystem given the same key. This is a property of a secure stream cipher such as xSalsa20 used for sealed-box (see XSalsa20).

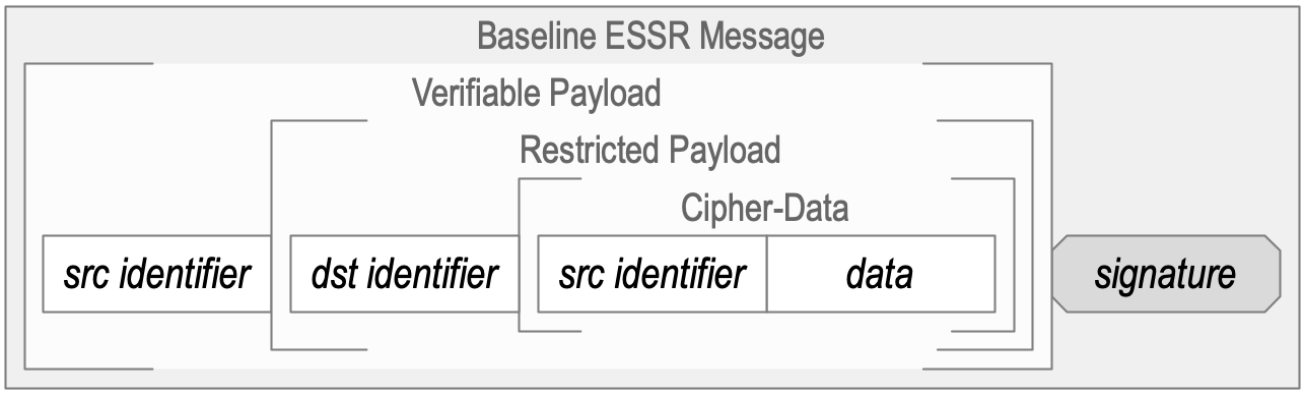
This is shown in the following table: (see ESSR)

scheme	TUF-PTXT	TUF-CTXT	RUF-PTXT	RUF-CTXT
Encrypt-then-Sign	X	X		X
ESSR	X	X	X	X

In summary, as a result of binding the sender's public key inside the ciphertext and binding the receiver's public key in the enclosing signed plain text an adversary is prevented from forging messages that compromise either authenticity or confidentiality. Thus ESSR provides both strong authenticity and strong confidentiality.

Baseline KERI ESSR

We apply ESSR to KERI by modifying the basic ESSR approach to support unbounded-term identifiers by replacing the sender's long-term public key with its AID and the receiver's long-term public key with its AID. The public keys are then securely looked up from the respective KELs of the sender and receiver. This is diagrammed below.



ESSRBaselineMessage

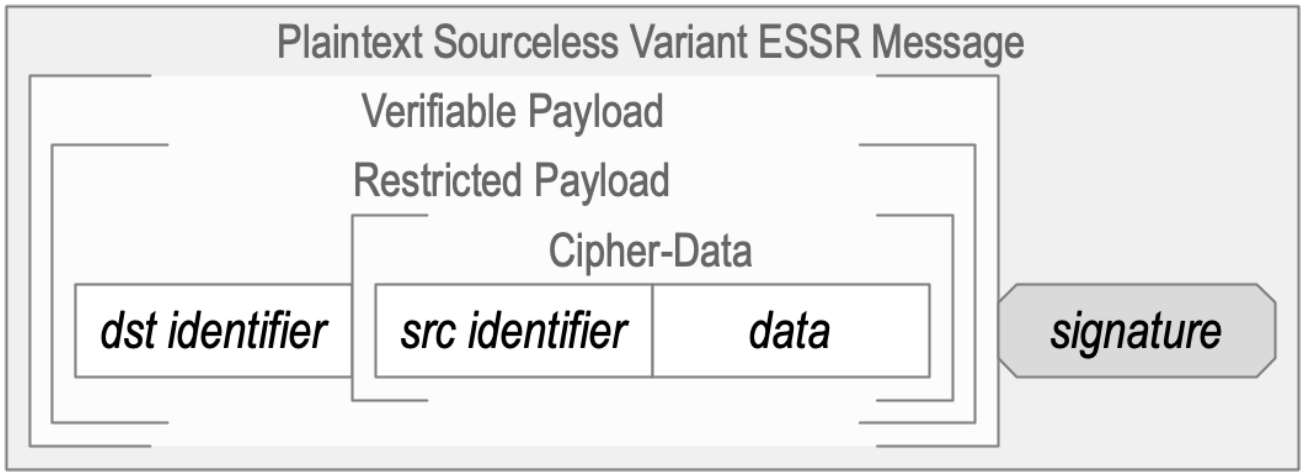
Diagram: Baseline ESSR Message

Plaintext Sourceless Variant

As shown in the diagram above, the sender's AID appears both in the verifiable payload plaintext and the restricted payload ciphertext portion. The first appearance in the plaintext portion is so that the receiver or any party can validate the signature without first decrypting the ciphertext. This is essential for upstream DDOS protection using load balancers and firewalls which do not have access to the receiver's private decryption key and therefore can't make a drop decision based on the sender via a whitelist or blacklist. To reiterate sharing the private decryption key with a load balancer or firewall upstream of the recipient in order to discover the sender's public verification key would break confidentiality.

Another limitation of removing the plain-text Sender AID is that the message's ciphertext is not publically verifiable. This means the authenticity of the ciphertext is not verifiable merely using public information. In some cases, a 3rd party may need to verify the authenticity of the ciphertext without seeing the plaintext from which the ciphertext was generated.

That said there may be some applications where upstream DDOS and/or firewall protection and/or public verifiability is not needed. Because the sender AID appears twice, once in plain text and once in ciphertext if DDOS protection or public verifiability is not needed then the plaintext appearance could be removed without harming authenticity. In that case, DDOS or firewall protection (security) could be traded for not exposing the sender's AID (privacy) without compromising the authenticity or confidentiality of the message. The following diagram shows the message without the sender's AID in plaintext only in the ciphertext.



ESSRPTSrclessMessage

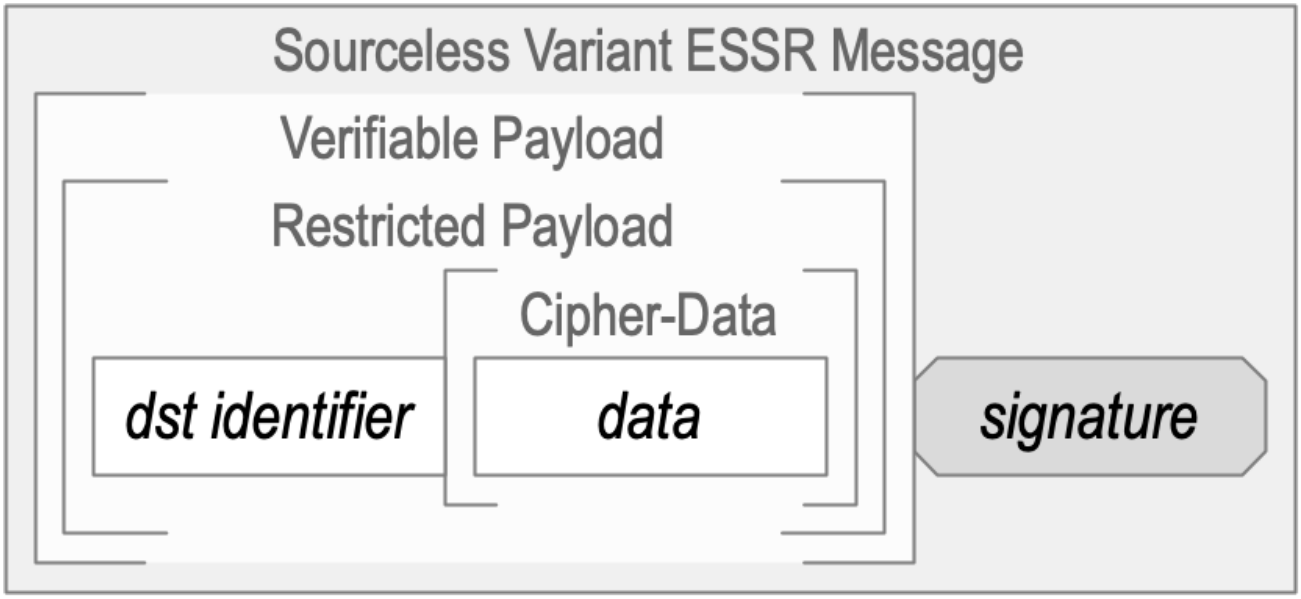
Diagram: Plaintext Sourceless Variant ESSR Message

In the case where the receiver follows the policy of only one relationship identifier (OORI) then it may be possible to maintain public verifiability without the sender AID in plaintext. With this policy, OORI, a given receiver AID, is only used for one given sender AID. The receiver can then store the sender's AID for each of its OORI AIDS. This means that the receiver can assume that there is only one possible sender for each of its receiver AIDs. As a result, the sender AID in the message plaintext may not be necessary. With an OORI policy, the receiver only accepts messages to a given receiver AID from a given sender AID that it looks up based on the receiver AID in the message plaintext. If the signature does not verify against the looked-up sender AID then the message is dropped. This unique mapping between receiver AID and sender AID can be shared upstream with a load balancer or intermediary so that they can do public verification of incoming messages. With an OORI policy, the message diagrammed above would be publically verifiable in spite of not providing the src identifier in plaintext.

Sourceless Variant

As described above, with an OORI policy, the receiver only accepts messages to a given receiver AID from a given sender AID. This policy may remove the need to include the sender's AID in the ciphertext as well. When a receiver follows the OORI policy, it means that any other sender besides the one associated with the receiver AID can not strip the signature for a given ciphertext to the given receiver AID and replace it with its own because the receiver will drop it if the signature does not correspond to the sender AID allowed for that

receiver AID. Notwithstanding the former, removing the ciphertext source AID, even with an OORI policy, still exposes the receiver to an attack when the approved sender is malicious. In this attack, the approved sender provides ciphertext for plaintext it has never seen but purports that it has seen it. Thus an OORI policy does not completely protect the receiver. But some applications may want to make this tradeoff. This variant is diagrammed below.



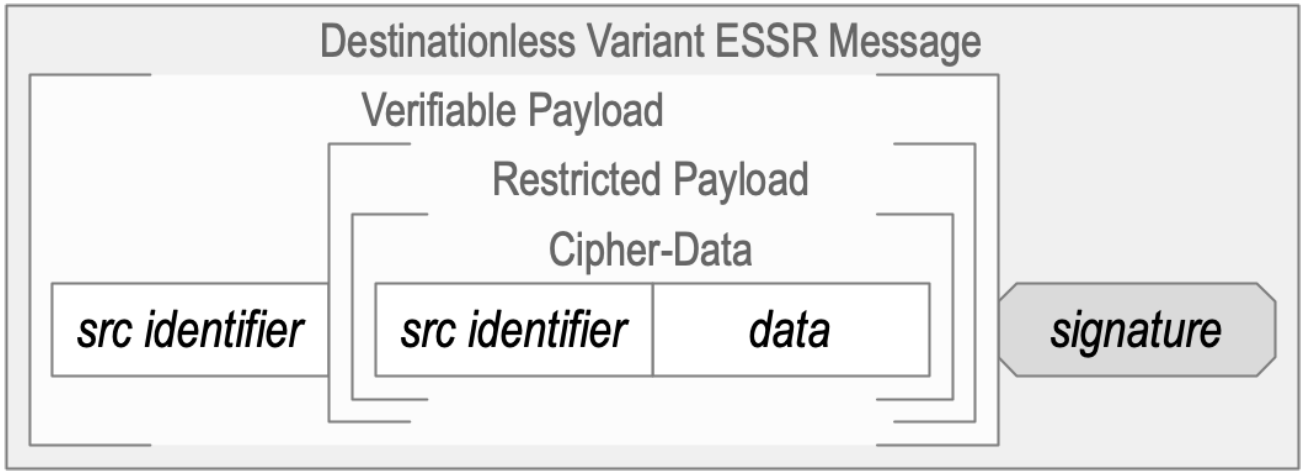
ESSRSrclessMessage

Diagram: Sourceless Variant ESSR Message

Destinationless Variant

A subtle advantage of using AIDs instead of public keys is that it may make receiver key compromise impersonation more difficult. This allows us to make modifications to the baseline ESSR approach. This is because the only valid receiver's keys are those in the receiver's KEL. An attacker can't substitute a different key, not in the KEL; it must use a stale key. To elaborate, using AIDs, an attacker can't purport any plaintext to have been encrypted to a different key pair other than the one that appears in the receiver's KEL. So any ciphertext must decrypt using one of the receiver's keypairs, either the current one or a stale one. A malicious receiver can't encrypt any plaintext with any keypair it chooses. The malicious receiver must either find a combination of stale key pair and plaintext that generates the same signed ciphertext as its current keypair or confuse the sender into signing ciphertext generated by the receiver with its current encryption keypair for plain text not yet seen by the sender.

Given that the only choices for key-pair are those in the receiver's KEL, the ability of a compromised receiver to generate any signed ciphertext using some combination of key-pair and plaintext is more limited. This looks like a type of birthday table attack BDay Attack. Given that the key choice is so limited, a malicious receiver may not gain any appreciable advantage in being able to construct the same ciphertext using a stale keypair as the ciphertext constructed with the current key pair but already signed by the sender. When that is the case (i.e., not vulnerable to BDay attack) and there is also a way for an honest receiver to look up the intended long-term public decryption key without having its AID in the message, then the destination AID may be removed from the message. This variant is shown in the diagram below.



ESSRDstlessMessage

Diagram: Destinationless Variant ESSR Message

Replay Attack Protection

Furthermore, adding a replay attack prevention mechanism against the reuse of the stale keys would provide additional protection against receiver KCI by both the sender and any third parties. A simple replay attack protection mechanism would be to adopt a policy that once the key state has been updated no messages signed with stale keys may be accepted. A more complicated replay attack protection mechanism would be for the sender to anchor a digest (content address) of the message in the sender's KEL. In this case, the signature attached to the message is not needed only a reference to the anchor in the sender's KEL.

KERI ESSR Summary

In summary, the baseline for the best practices for message security follows the full ESSR message policy shown in the diagram above (labeled Authenticatible Confidential Message). Any variation from the full ESSR policy may make a security trade-off induced by removing one of its protections. This trade-off should be explicated in the protocol design so as to inform its proper use.

Relationships

Basics

The final preliminary before directly addressing privacy is to define what we mean by relationships. Suitable relationship properties will give us a handle on managing privacy with respect to exploitably correlatable identifiers. We define a relationship as a two-tuple (duple) or pairing between two AIDs. The pairing may also have a direction (bi-directional or uni-directional). Thus for any two AIDs there are three different relationships. Let `A` and `B` denote each aid and let `<>`, `->`, `<-`, and denote directionality. Then we can represent each of the three relationships as follows:

```
(A, <>, B)
```

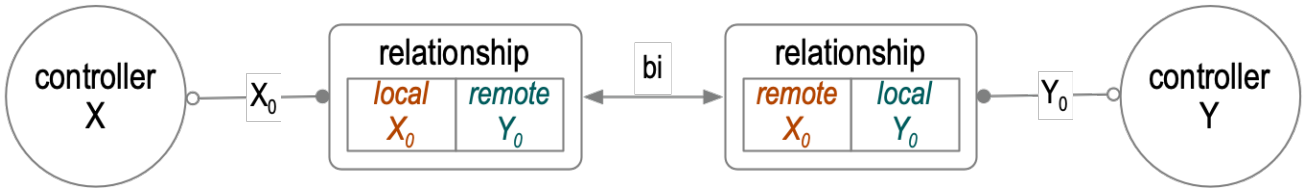
```
(A, ->, B)
(A, <-, B)
```

The first one above is bi-directional, the second is uni-directional from `A` to `B`, and the third is uni-directional from `B` to `A`.

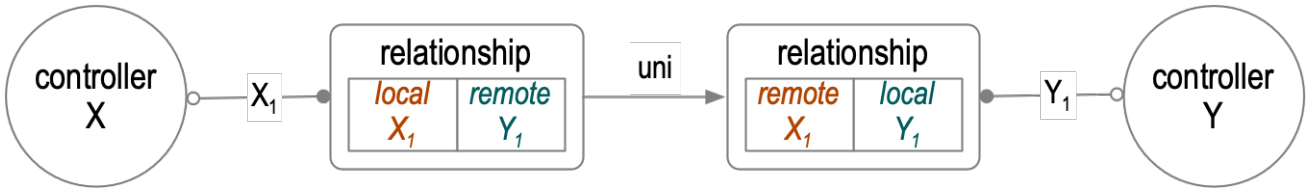
We also use the convention that a given party that controls an AID in a relationship always puts its own AID as the first AID. This we call the `(local, remote)` convention. When party `A` always denotes itself as `local` with respect to its relationship with `B` as `(A, ->, B)` and party `B` denotes itself as local with respect to its relationship with party `A` as `(B, <-, A)`. This convention makes the implementation of relationship graphs consistent in that they are always directed from the perspective of the `local` party.

When a given party has multiple AIDs then we can compactly denote those by subscripts. Say party *X* has three different AIDs we denote them X_0, X_1, X_2

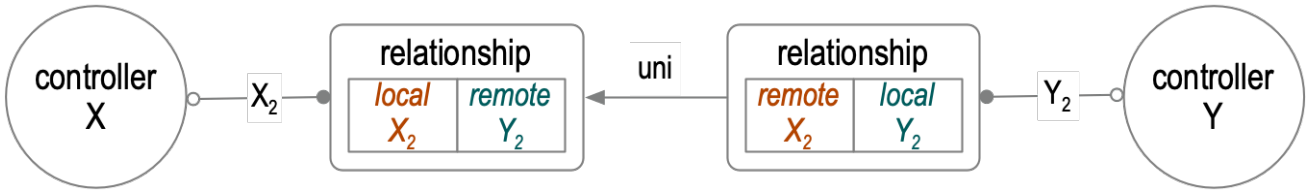
Diagrams of showing the relationship conventions are shown below:



RelationshipBi



RelationshipUniDiOut



RelationshipUniDiIn

Diagram: Relationships

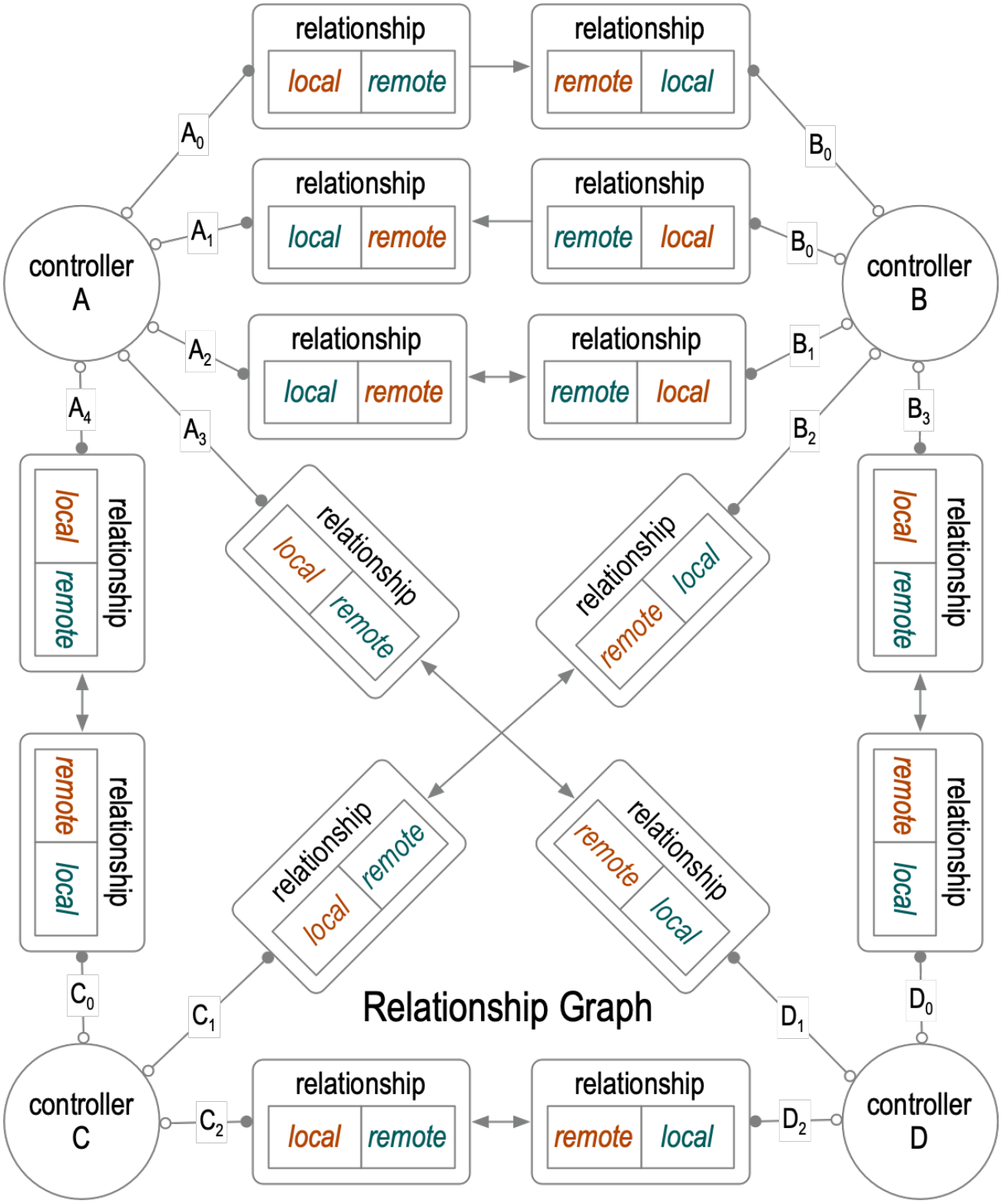
Now that we have introduced the concept of a relationship, we formally define its properties:

- A cryptonym is a cryptographically derived pseudonym with at least 128 bits of entropy in the derivation.
- An AID is a cryptonym that is also securely attributable to one or more key pairs.
- A relationship is a pairing of two AIDs, one each from a different controller.
- Two different AIDs are by themselves uncorrelated in an information-theoretic security sense in that knowledge of one by itself provides no information about the other.
- A context is a set of events.

- Two contexts are disjoint with respect to an AID when that AID appears in one or more events in one set but does not appear in any event in the other set.
- A relationship is not a communication channel, but the events sent over a communication channel may be a context for the aids in a relationship.
- A partition is a set of contexts with mutually disjoint relationships.
- Any set of relationships using ORIs (One Relationship Identifiers) may form a partition.
- Any two members (contexts) of a partition may be correlatable due to other information associated with any pair of contexts, but the partitioned relationships by themselves provide no correlatable information, i.e., partitioned relationships are by themselves not correlatable
- An AID that is common to any subset of events within a context provides a perfectly correlatable feature across those common events in that context.
- The secure attributability of an AID, together with its perfect correlatability within a context, enables secure reputational trust (good or bad) in that AID within that context
- partitions balance the concerns of:
    - strong authenticity and strong confidentiality within a context with,
    - sufficiently strong privacy between contexts.

Relationship Graphs

Given the definition of a relationship, we can define a relationship graph. Each controller can manage its AIDS and the associated relationships with AIDs from other controllers. This is shown below for 4 controllers denoted *A*, *B*, *C*, *D*. The graph is all-inclusive, showing both sides of each relationship for all controllers, but a given controller may, in actuality, only see a partial version of the graph.

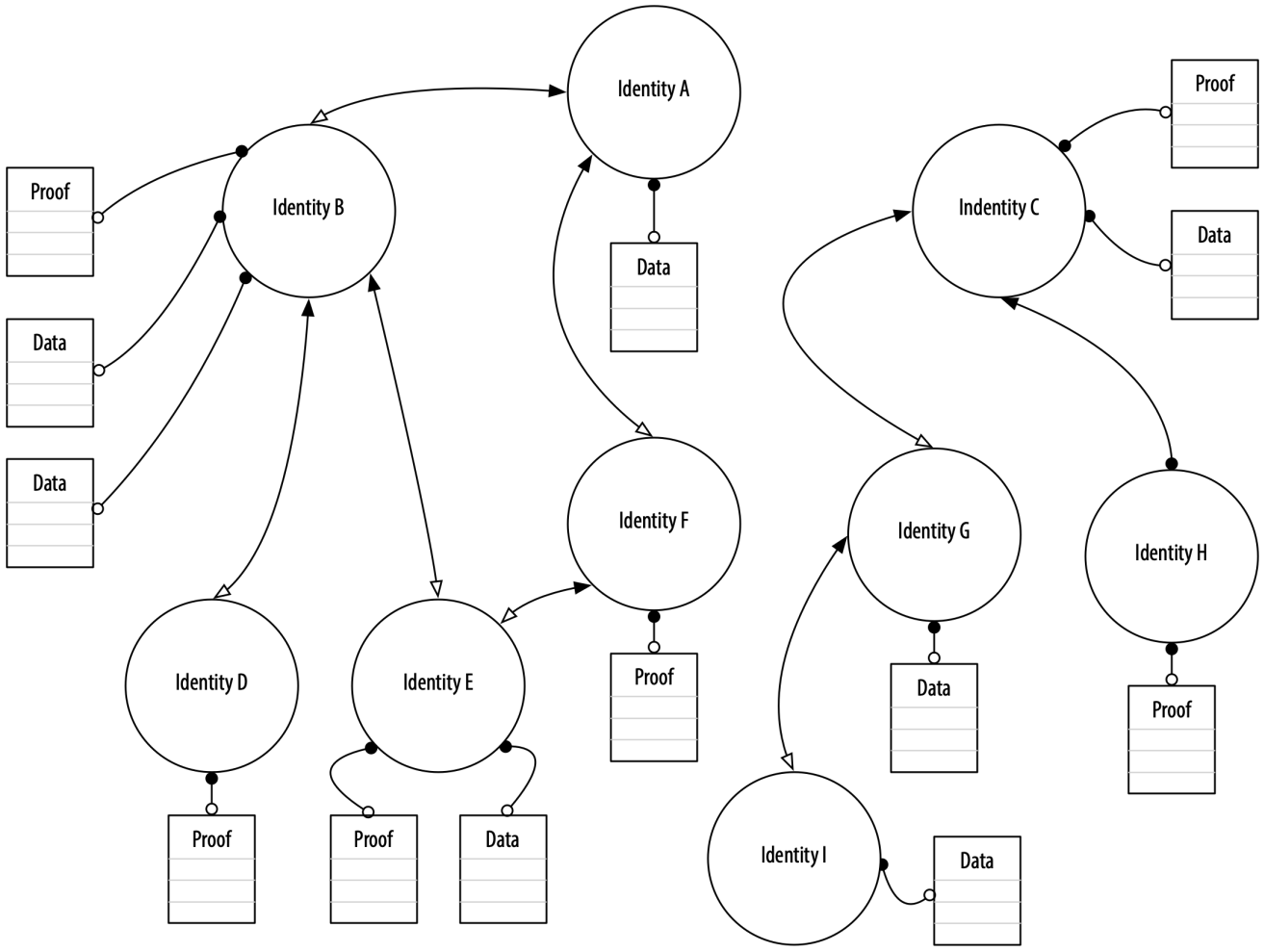


RelationshipGraph

Diagram: Relationship Graph

A relationship graph is actually a subset of what I call an identity or, more accurately, an identifier graph. A relationship graph is the intersection of the identifier graphs for multiple controllers. An identifier graph shows the correlatable connections between the identifiers that a given controller uses for itself and the identifiers other controller use for themselves. The theory of identifier graphs (and relationship graphs as a subset) was

first explicated in the Open Reputation whitepaper (circa 2015). It has been prominently mentioned since then in various presentations (primarily on reputation systems) and other whitepapers identity system essentials. A self-identity graph is shown below:



IdentityGraph

Diagram: Self-Identifier Graph

Given a relationship graph with partitioned relationships (AIDs used in partitioned contexts), a given controller can isolate the correlatability of those AIDs. We now have the preliminaries necessary to start exploring how to design protocols that, as a point of departure, make no compromises on authenticity and confidentiality but provide sufficient privacy by effectively mitigating exploitation due to identifier correlation via relationship partitions.

Syntax

In order to more efficiently represent the messages or packets for any given protocol we adopt the following textual syntactical conventions.

AIDs

AIDs from a controller are denoted by a capital letter representing the controller with a unique subscript for each AID. For example, the controller *X* may have AIDS as follows:

X_0, X_1, X_2

Signatures

The content signed by a signature is delimited with angle brackets, <> surrounding the data being signed, and is followed by the AID of the signer, which denotes the attached signature itself. For example,
$< data >X_0$,
means *data* is signed with the attached signature using the current signing keys for AID, X_0. When more than one signature is attached then the signatures are delimited by a comma-separated list surrounded by parenthesis as follows:

$< data >(X_0, X_1)$.

Ciphertext

Ciphertext is delimited with curly brackets surrounding the ciphertext and is followed by the AID, whose current en-de-cryption keypair provides the public encryption key and private decryption key. For example,

$\{ data \}Y_0$,

means *data* is ciphertext encrypted to Y_0. Fields within cipher text are comma-separated and may be grouped using parenthesis.

Messages

A message is delimited by square brackets surrounding the components. Fields may be comma separated and may be grouped with parenthesis. Nested messages are delimited by nested square brackets. For example,

$[src\ X_0, dst\ Y_0, data]$,

denotes a message with three plaintext fields; the first is the src address given by the AID X_0, the second is the dst address given by Y_0, the third is a data field.

A more complicated example is as follows:

$<[src\ X_0, dst\ Y_0, \{src\ X_0, data\}Y_0]>X_0$,

which denotes a message signed by X_0 with embedded ciphertext encrypted to Y_0 which ciphertext includes src and data. The message also includes plaintext fields for the src and dst addresses. This is an example of a message in full ESSR format.

Header

Any practical protocol requires a header with one or more header fields in plaintext that provide metadata about the packet. This metadata typically includes version, protocol type, and packet type, and depending on the protocol, may include service type and interaction state support like interaction ID, sequence number, or hash references to other packets in a transaction. For security, this header should be protected by being placed in the signed portion of any packet. The protected header enables a parser to interpret the packet without exposing it to attack. A practical version of the example above would include such a header as shown below:

<[header, src X_0, dst Y_0, {src X_0, data}Y_0]>X_0

However, for the purposes of understanding how to analyze the trade-offs of the fundamental properties of authenticity, confidentiality, and privacy, the header details are largely irrelevant. Without loss of generality, then, it is assumed that any practical packet will have a header with whatever additional metadata is required by the specific protocol. But for the purposes of this exposition, we will simplify the message descriptions by leaving out the header metadata unless it is relevant. We will return to the topic of how to manage transaction or interaction-specific correlatable meta-data in a later section.

Transport (IP) Addresses and Header

In order to route a message over the internet, the packet must have an IP (UDP or TCP) header with a *src* (source) address and port and a *dst* (destination) address and port. We assume that the IP header is not protected by the signature. This assumption is still valid for other transports besides IP. We assume that there will be a transport header with *src* and *dst* addresses or, more generally, *src* and *dst* service endpoints. Therefore without loss of generality, we simply label the transport endpoints with the modifier *ip*. We also assume that for each AID, there may be default corresponding transport endpoints for the source (*src*) and the destination (*dst*). Of course, a given AID may have multiple endpoints, and this may be distinguished with a *role* label, but this is a management detail that does not impact the primary issue of correlation.

Suppose that for AID, X_0, we denote its transport source endpoint as *src ip* X_0, and we denote its destination transport endpoint as *dst ip* X_0. These transport source and destination addresses may appear in a message in its protected (signed) metadata besides appearing in the unprotected (unsigned) IP header as needed for different virtual routing purposes.

An unprotected transport header (ip) is delimited with vertical brackets, //. Thus for a packet between source AID X_0 and destination AID Y_0 the transport header is given by,

|src ip X_0, dst ip Y_0|.

An example of a full packet with prepended transport header is shown below,

|src ip X_0, dst ip Y_0|<[src X_0, dst ip Y_0, dst Y_0, {src X_0, data}Y_0]>X_0

Please note that in this particular example, the destination transport (*ip*) address appears twice. Once in the unprotected transport header and once more in the protected message body.

Unlike an AID, an IP address is public by nature. This means each IP address comes with an inherently correlatable global context. It is, therefore, not private by default. IP addresses are assigned largely on a geographic basis, so the mere appearance of an address provides potentially exploitable correlatable information. IP addresses must appear in plaintext in the IP header in order for the associated IP packet to be routable over the internet. This appearance inherently correlates sources to destinations. As a result, the challenge for mitigating exploitation by aggregators of identifiers starts with mechanisms that mitigate the correlation of the IP addresses first, not the AIDs. Indeed, as we shall see below. Mechanisms that mitigate IP address correlation may, with little or no extra effort, also mitigate AID correlation.

OOB Setup

Any secure over-the-wire protocol on a given communications channel between two parties requires at least one out-of-band-authentication (OOBA) factor in order to protect against a man-in-the-middle (MITM) attack. The OOBA is out-of-band (OOB) with respect to the communications channel. The OOBA does not have to be directly between the two parties but may involve a third party. There just needs to be information available to the two parties that they obtain out-of-band with respect to the MITM so that the MITM does not have access to that information.

This means that without loss of generality, we can make the assumption that every protocol has at least one OOB setup. To simplify the discussion we can assume that the two parties exchange identifiers OOB. We can always complicate things by having more elaborate OOB setups. But this does not change the requirement that at least one authenticating factor happens OOB i.e., an out-of-band-authentication (OOBA).

Requiring at least one OOBA factor in the setup solves a second problem. This problem is called *cheap pseudonymity*. Cryptonymous identifiers may be created on the fly. Because they are derived from high entropy capture (i.e., are long strings of pseudorandom characters), there are a practically infinite number of them. This makes them cheap. So any protocol that wants to be protected from DDOS, where a malicious entity just creates a new identifier in order to bypass a filter, needs some way to manage *cheap pseudonymity*. Requiring an OOBA factor for any given identifier makes that identifier relatively expensive. An attacker can’t create a replacement without repeating the OOBA for that new identifier.

Finally, an OOBA (out-of-band-authentication) may be simultaneously strongly authentic, confidential, and private. This provides a secure basis for in-band transmission without yet compromising authenticity, confidentiality, or privacy.

As a result, we will assume an initial OOBA setup that is an exchange of identifiers as the default departure point for protocol design. This assumption does not preclude other setups or other setup mechanisms. One can imagine two one-way OOBAs separated in time and space, or where one side has a public AID that has been previously reputed using some 3rd-party OOBA mechanism like a registry. Any other setup may then be analyzed by comparison for the trade-offs that it makes with respect to security in terms of authenticity, confidentiality, privacy, DDoS protection, friction, etc. The fundamental idea is that the AIDs of the two parties at the ends (edges) of a conversation must be discovered somehow in an OOB fashion in order to protect against MITM and cheap pseudonymity. How that discovery happens may be highly use-case specific.

Baseline Protocols

Direct

Suppose we have a pair of entities denoted by AIDs *A* and *B*. An OOB exchange of identifiers provides to entity *A*, *B*'s' AID B_0 and *B*'s service endpoint *ip* B_0, it also provides to entity *B* with *A*'s' AID A_0 and *A*'s service endpoint *ip* A_0. *A* and *B* form a bidirectional relationship $(A_0, <>, B_0)$. The context of this relationship is direct communication between *A* and *B*

An authenticatable message from *A* to *B* with confidential content data using ESSR is denoted as follows:

$|src\ ip\ A_0, dst\ ip\ B_0|<[src\ A_0, dst\ B_0, \{src\ A_0, data\}B_0]>A_0$

Similarly authenticatable message from *B* to *A* with confidential content data using ESSR is denoted as follows:

$|src\ ip\ B_0, dst\ ip\ A_0|<[src\ B_0, dst\ A_0, \{src\ B_0, data\}A_0]>B_0$

A variant where the plaintext is also signed when required by legal recourse is a follows:

$|src\ ip\ A_0, dst\ ip\ B_0|<[src\ A_0, dst\ B_0, \{<src\ A_0, data>A_0\}B_0]>A_0$

$|src\ ip\ B_0, dst\ ip\ A_0|<[src\ B_0, dst\ A_0, \{<src\ B_0, data>B_0\}A_0]>B_0$

The IP addresses and AID are correlatable by any ISP or intermediary that watches the messages since they all appear in plaintext. But the confidential content data is not observable. The only correlatable information is that A_0 is talking to B_0 using IP endpoints *ip* A_0 and *ip* B_0. This may be monetizable by an ISP for advertising aggregation. On the other hand, because it is strongly authentic and confidential, it, by itself, is not very exploitable by an identity thief.

So it's strongly authentic and confidential but not private.

Now suppose that *A* and *B* want to communicate about a business deal using a relationship that is unique to the business deal context but not the communication context.
For that business deal context they use the relationship $(A_1, <>, B_1)$. Since this context is not used for communication it does not have any associated IP addresses. Instead, it must be transported over a communication context relationship.

Suppose there is an OOB exchange of A_1 and B_1 between *A* and *B*. Now *A* and *B* can communicate totally authentically, confidentially, and privately using the relationship associated with the business deal context but leveraging the communications context relationship. These are shown as follows:

From *A* to *B*:

$|src\ ip\ A_0, dst\ ip\ B_0|<[src\ A_0, dst\ B_0, \{src\ A_0, <[src\ A_1, dst\ B_1, data]>A_1\ \}B_0]>A_0$

Embedded in this confidential message is an authenticatable message of content data from A_1 to B_1 and signed by A_1.

Similarly from *B* to *A*:

|src ip B_0, dst ip A_0|<[src B_0, dst A_0, {src B_0, <[src B_1, dst A_1, data]>B_1 }A_0]>B_0

Embedded in this confidential message is an authenticatable message of content data from B_1 to A_1 and signed by B_1.

No 3rd party observer can view that there is a relationship *(A_1, <>, B_1)* via the communication's context messages. Thus there is no leakage via correlation to any other use of the *(A_1, <>, B_1)* relationship. The *(A_1, <>, B_1)* relationship is a no-compromise, authentic, confidential, and private relationship between *A* and *B*.

To elaborate, *A* and *B* now have two different relationship contexts. These are mutually partitioned as far as any 3rd party observer is concerned. The first, *(A_0, <>, B_0)* is the communication context. It may be thought of as a *hop-wise* communication context because there is only one hop at the AID level of routing between A_0 and B_0. At the IP level, there may be multiple hops. The second, *(A_1, <>, B_1)*, is the interaction context associated with the business deal that may span multiple messages. It may be thought of as an *end-wise* interaction context at the AID level.

Relationship Formation Protocol

Instead of using another OOB setup to form *(A_1, <>, B_1)*, we can leverage the OOBA setup used to form the pre-existing communications context relationship *(A_0, <>, B_0)*. Essentially, *(A_0, <>, B_0)* acts as a 3rd party reference or reputable letter of introduction to form *(A_1, <>, B_1)*. This suffices to protect against MITM attacks and cheap pseudonymity. We just need a new protocol type with message types for forming a new relationship. This new protocol could include a relationship formation initiation message that could have semantics something to the effect of:

I, the controller of A_0 wish to establish a new relationship using A_1 with the controller of B_0. This would be signed by A_1 as well as the A_0 signature wrapping the message.

Then the controller of B_0 can either refuse the initiation with a NACK or can accept with a relationship formation acceptance message that could have semantics something to the effect of:

I the controller of B_0 accept a new relationship using B_1 with the controller of A_1. This would be signed by B_1 as well as the B_0 signature wrapping the message.

The accept message can include a digest (SAID) of the initiation message, which digest could be used as the interaction ID. Then the AIDs in the initiation message do not need to be repeated in the accept message. Adding a nonce to the first message (initiation) means that its context cannot be guessed using a rainbow table attack which could be used when logging interactions. When the digest of the first message has high entropy, then so does the digest of each subsequently chained message in a sequence of digest (hash) chained messages.

Such hash chaining of multi-message protocols where the messages are also signed makes the set of messages in the interaction a verifiable data structure. This is virtually impervious to attack. Moreover, because cryptographic digests are universally unique, using a digest for the interaction ID of a chain of hash-chained

digests that form an interaction chain enables a given interaction to span different communication contexts and still be easy to lookup in an interaction database that is indexed by such digests. Finally, this approach makes it easy to detect replay attacks.

The relationship formation protocol defined above would allow any pre-existing relationship to bootstrap a new relationship without having to perform yet another OOB setup to exchange identifiers. This could also be used to establish new communications context relationships by adding the exchange of the service endpoint IP addresses in addition to the AIDs for the newly formed relationship. This is all accomplished using a no-compromise secure authentic, confidential, and private exchange between the two parties.

Thus, any given relationship that has been sufficiently authenticated and is therefore reputable (not cheap) can be used to issue the equivalent of a letter of introduction for a new relationship, thereby imbuing the new relationship with reputation. We call this an Out-Of-Band-Introduction (OOBI) because the reputable relationship authentication was established Out-Of-Band with respect to the new relationship.

The original communication context relationship does not leak any correlatable information about the newly formed relationship. The only correlatable information is that each communication context relationship AIDs are correlated with the relationship's IP addresses. To reiterate, There is no correlation to any other relationship, including other communication relationships, unless they both use the same or related IP addresses.

Therefore if we can solve ip address correlation we have solved 3rd party exploitability of identifiers for advertising aggregation. This we show in the following section.

One Shared Intermediary

The direct messaging protocol above does not provide correlation privacy with respect to identifiers used in the communications context relationship $(A_0, <>, B_0)$ between *A* and *B*. Both the IP addresses and the AIDs or in plaintext and may be correlated by either of the ISPs for *A* and *B*. This section will detail how a single shared intermediary may be used to provide privacy for the AIDS and ip addresses associated with a communications context relationship between *A* and *B*. The intermediary may provide sufficient herd privacy so that the communications between *A* and *B* are largely unexploitable for advertising aggregation purposes.

Let the intermediary controller be denoted by *C*.

Suppose *A* forms a communication context relationship with *C* given by $(A_2, <>, C_0)$ with ip endpoints $ip\ A_2$ and $ip\ C_0$. The initial exchange of these identifiers is performed OOB so that the initial setup of the relationship is securely authentic, confidential, and private.

Suppose *B* forms a communication context relationship with *C* given by $(B_2, <>, C_1)$ with ip endpoints $ip\ B_2$ and $ip\ C_1$. The initial exchange of these identifiers is performed OOB so that the initial setup of the relationship is securely authentic, confidential, and private.

Further, suppose that *C* has communication relationships with a large number of other entities (say > 1000).

We assume that *A* and *B* have formed at least one relationship by exchanging identifiers. This could have been formed using their direct communications relationship or using some OOB exchange of identifiers. For the sake of convenience let that relationship be denoted $(A_1, <>, B_1)$. Now *A* and *B* can communicate using this

relationship in a private (non-correlatable identifier) manner by leveraging each of their communication's relationships with C. This requires that *A* and *B* also exchange OOB the AIDs of the intermediary's side of their respective communication's relationships with *C*. In this case *A* also knows C_1 and *B* also knows C_0.

Further, *C* maintains a database that maps the AIDs of each of its communications relationships to the corresponding ip service endpoints. this allows *C* to look up the current service endpoint given an AID. So for example *C* has the following signed entries in its database:

$<[C_0,<[A_2, ip\ A_2]>A_2]>C_0$
$<[C_1,<[B_2, ip\ B_2]>B_2]>C_1$

This makes the database zero-trust because it is signed at rest. An attacker can't forge or impersonate the ip service endpoint for any given relationship nor impersonate the AID of an associated relationship and thereby fool the intermediary into misdirecting traffic to the wrong endpoint or AID or to the right endpoint but using the wrong relationship (either of these would be the equivalent of a DNS hijack).

From *A* to *B* via *C*:

First hop from *A* to *C*:

$|src\ ip\ A_2, dst\ ip\ C_0|<[src\ A_2, dst\ C_0, \{src\ A_2, dst\ C_1, <[src\ A_1, dst\ B_1, \{src\ A_1, data\}B_1]>A_1\ \}C_0]>A_2$

Upon receipt of this message *C* verifies the source as coming from one of its authenticated relationships. In this case A_2. Then *C* can extract the destination C_1 to lookup first B_2 and then the associated dest *ip* B_2.

Last hop from *C* to *B*:

Finally *C* forms a new message to *B* as follows:

$|src\ ip\ C_1, dst\ ip\ B_2|<[src\ C_1, dst\ B_2, \{src\ C_1, <[src\ A_1, dst\ B_1, \{src\ A_1, data\}B_1]>A_1\ \}B_2]>C_1$

Now *B* can verify that this came from C_1 and then decrypt the contents which it can then verify as coming from A_1. No 3rd party every sees the identifiers from $(A_1, <>, B_1)$. Therefore this relationship is private with respect to correlatable identifiers.

All any 3rd party ISP sees is that A_2 has a relationship with *C* and B_2 has a relationship with *C* but can't correlate those relationships to $(A_1, <>, B_1)$ given sufficient herd privacy of *C*'s relationships with others. This provides secure authenticity, confidentiality and privacy between $(A_1, <>, B_1)$ that only requires that *A* and *B* trust *C* not to reveal that $(A_1, <>, B_1)$ exists.

Note that the content data from A_1 to B_1 is not seen by *C*, only the AIDs. Moreover, the AIDS in $(A_1, <>, B_1)$ are never stored on disk by *C*. They are only ever in memory. This means that an attacker who compromises the disk storage of *C* cannot leak $(A_1, <>, B_1)$. An attack on *C*'s protected memory processes is much more difficult than an attack on *C*'s disk storage. This is more than sufficient to protect against 3rd party correlation for advertising aggregation.

Furthermore, note that *A* does not need to know the *B* side of *B*'s relationship with *C*, and *B* does not need to know the *A* side of *A*'s relationship with *C*. So neither can leak that information. An attacker has to get that by attacking *C*, not either *A* or *B*.

From a relationship context perspective, *A* and *B* have isolated their independent *hop-wise* communication contexts with *C*, namely, *(A2, <>, C0)* and *(B2, <>, C1)*, from their joint *end-wise* context, namely, *(A1, <>, B1)*. Given that there are now two hops, the end-wise relationship serves two purposes. The first is an end-to-end routing context at the AID level. The second is as an end-to-end interaction context for their business interaction that may span multiple messages. Importantly these three contexts (independent hop-wise and joint end-wise) are mutually partitioned as far as any 3rd party observer is concerned.

The primary limitations of this approach are as follows:
* Both *A* and *B* must trust *C*. This may be problematic in general.
* The ISP of *C* has access to both the source and destination IPs of all *C*'s packets. With some extra work, the ISP could correlate observable statistics (not identifier per se) about the packets such as time-of-arrival and time-of-departure, packet size, packet type, protocol types, etc. Given enough traffic between *A* and *B*, this might be enough for the ISP to get some degree of correlation. It is unlikely, however that such a weak correlation would be monetizable by the *C*'s ISP for advertising aggregation.

In the next section, we solve both of the limitations above by using two intermediaries. One trusted by *A* and one trusted by *B*. As long as both intermediaries do not share the same ISP then there would not even be a weak correlation between *A* and *B* solely based on their joint traffic. This, of course, assumes sufficient herd privacy for each of the intermediaries.

Non-Shared Intermediaries

This differs from the previous protocol in that *A* and *B* each has an intermediary that they each trust. Let *C* denote *A*'s intermediary and *D* denote *B*'s intermediary.

Suppose *A* forms a communication context relationship with *C* given by *(A2, <>, C0)* with ip endpoints *ip A2* and *ip C0*. The initial exchange of these identifiers is performed OOB so that the initial setup of the relationship is securely authentic, confidential, and private.

Suppose *B* forms a communication context relationship with *D* given by *(B2, <>, D0)* with ip endpoints *ip B2* and *ip D0*. The initial exchange of these identifiers is performed OOB so that the initial setup of the relationship is securely authentic, confidential, and private.

Further, suppose that *C* and *D* each has independent communication relationships with a large number of other entities (say > 1000).

Further, both *C* and *D* maintain a database that maps the AIDs of each of its communications relationships to the corresponding ip service endpoints. this allows each to look up the current service endpoint given an AID.

For example *C* has the following signed entry in its database:

<[C0,<[A2, ip A2]>A2]>C0

Likewise, *D* has the following signed entry in its database:

$<[D_0,<[B_2, ip\ B_2]>B_2]>D_0$

This makes the databases zero-trust because there are signed at rest. An attacker can't forge or impersonate the ip service endpoint for any given relationship nor impersonate the AID of an associated relationship and thereby fool the intermediary into misdirecting traffic to the wrong endpoint or AID or to the right endpoint but using the wrong relationship (either of these would be the equivalent of a DNS hijack).

We assume that *A* and *B* have formed at least one relationship by exchanging identifiers. This could have been formed using their direct communications relationship or using some OOB exchange of identifiers. For the sake of convenience let that relationship be denoted $(A_1, <>, B_1)$. Now *A* and *B* can communicate using this relationship in a private (non-correlatable identifier) manner by leveraging each of their communication's relationships with their respective intermediaries, *A* with *C* and *B* with *D)*. *This requires that* A* and *B* also exchange OOB the AIDs of the intermediary side of their respective communication's relationships with their own intermediaries. In this case *A* also knows D_0 and *B* also knows C_0.

Finally, in order for *C* and *D* to communicate with each other, they must both have an AID and corresponding ip service endpoint that will send and receive authenticated traffic from other intermediaries. If this is public then there is no need for an OOB exchange of this information. If not then *A* and *B* must also exchange the cross-intermediary AID and ip service endpoint of each of their intermediaries. Lets *C*'s be denoted C_2 with $ip\ C_2$ and *D*'s be denoted D_2 with $ip\ D_2$. Now any previously unauthenticated intermediary is indistinguishable from a DDOSing cheap pseudonym with respect to a given intermediary. Therefore some form of reputation is required. One approach would be for *A* to authorize (D_2, $ip\ D_2$) to *C* and for *B* to authorize (C_2, $ip\ C_2$)* to *D*. Since *A* is already reputable from *C*'s standpoint and *B* is already reputable from *D*'s standpoint. A signed letter message acting as a letter of reference should suffice.

As a result *C* will have a database of approved intermediaries with the following entry:

$<[A_2, D_2, ip\ D_2]>A_2$.

Intermediary *C* is thereby DDOS protected and can whitelist any messages from D_2. Moreover, *C* has the forwarding $ip\ ip\ D_2$ for any messages destined for *D*.

Likewise, *D* will have a database of approved intermediaries with the following entry:

$<[B_2, C_2, ip\ C_2]>B_2$.

Intermediary *D* is thereby DDOS protected and can whitelist any messages from C_2. Moreover, *D* has the forwarding $ip\ C_2$ for any messages destined for *C*.

Effectively these database entries create a communication context relationship between *C* and *D* denoted, $(C_2, <>, D_2)$.

Given this setup, we can now send a fully secure authentic, confidential, and private message from *A* to *B* for relationship $(A_1, <>, B_1)$ through intermediaries *C* and *D*.

From *A* to *B* via *C* and *D*:

First hop from *A* to *C*:

|src ip A_2, dst ip C_0|<[src A_2, dst C_0, {src A_2, dst D_2, dst D_0, <[src A_1, dst B_1, {src A_1, data}B_1]>A_1}C_0]>A_2

Upon receipt of this message *C* verifies the source as coming from one of its authenticated relationships. In this case A_2. Then *C* can extract the nearside intermediary destination D_2 to lookup *ip* D_2 as well as the farside intermediary destination D_2.

Next hop from *C* to *D*:

Next *C* forms a new message to *D* as follows:

|src ip C_2, dst ip D_2|<[src C_2, dst D_2, {src C_2, dst D_0, <[src A_1, dst B_1, {src A_1, data}B_1]>A_1}D_2]>C_2

Now *D* can verify that the message comes from an approved source C_2 and then decrypt to extract the farside destination D_0. Given D_0, *D* can lookup B_2 and ip B_2].

Last hop from *D* to *B*:

Finally *D* forms a new message to *B* as follows:

|src ip D_0, dst ip B_2|<[src D_0, dst B_2, {src D_0, <[src A_1, dst B_1, {src A_1, data}B_1]>A_1 }B_2]>D_0

Now *B* can verify that this came from D_0 and then decrypt the contents which it can then verify as coming from A_1. No 3rd party every sees the identifiers from *(A_1, <>, B_1)*. Therefore this relationship is private with respect to correlatable identifiers.

All any 3rd party ISP sees is that A_2 has a relationship with *C* and B_2 has a relationship with *D* but can't correlate those relationships to *(A_1, <>, B_1)* given sufficient herd privacy of *C*'s and *D*'s relationships with others. This provides secure authenticity, confidentiality and privacy between *(A_1, <>, B_1)* that only requires that *A* trust *C* that *C* trust *A* and *B* trust *D* and *D* trust *B*. *A* and *B* both trust that *C* and *D* do not reveal that *(A_1, <>, B_1)* exists.

Note that the content data from A_1 to B_1 is not seen by either *C* or *D*, only the AIDs. Moreover, the AIDS in *(A_1, <>, B_1)* are never stored on disk by either *C* or *D*. They are only ever in memory. This means that an attacker who compromises the disk storage of *C* or *D* cannot leak *(A_1, <>, B_1)*. An attack on *C*'s or *D*'s protected memory processes is much more difficult than an attack on *C*'s or *D*'s disk storage. This is more than sufficient to protect against 3rd party correlation for advertising aggregation.

Furthermore, note that *A* does not need to know the *B* side of *B*'s relationship with *D*, and *B* does not need to know the *A* side of *A*'s relationship with *C*. So neither can leak that information. An attacker has to get that by successfully attacking both *C* and *D*. To elaborate, an attack on *A* and or *C* does not reveal the *B* side of *B*'s relationship with *D*. Both *C* and D must be attacked. Likewise, an attack on *B* and or *D* does not reveal the *A* side of *A*'s relationship with *C*. Both *C* and *D* must be attacked. Likewise, *A* does not have knowledge of *B*'s side of *B*'s relationship with *D*, only the *D* side. As well,

An attack on *C*'s disk does not expose that *A* has a relationship with any of *D*'s relationships because the farside destination is only exposed in memory. Likewise, an attack on *D*'s disk does not expose that *B* has any relationships with *A* because the farside destination is only exposed in memory.

This approach is fully zero-trust because all table lookups by *C* and *D* are signed at rest, so an attacker can't misdirect traffic between the two or between their associated edge relationships.

From a relationship context perspective, *A*, *B*, *C*, and *D* have isolated their independent *hop-wise* communication contexts with each other from the *A* and *B* joint end-wise context. The hop-wise contexts are *(A2, <>, C0)*, *(C2, <>, D2)*, and *(B2, <>, D0)*. The joint end-wise context is *(A1, <>, B1)*. Given that there are now three hops, the end-wise relationship serves two purposes. The first is an end-to-end routing context at the AID level. The second is as an end-to-end interaction context for their business interaction that may span multiple messages. Importantly these four contexts (independent hop-wise and joint end-wise) are mutually partitioned as far as any 3rd party observer is concerned.

We can conclude that we can have secure authenticity, confidentiality, and privacy (identifier correlation), as well as DDOS protection. The only OOB setups are one-time setups between *A* and *B*, *A* and *C*, and *B* and *D*. Indeed if there is a trust anchor (registry) that provides reputation to intermediaries, then there would be no need for *A* and *B* to provide letters of reference to their own intermediary for the other's intermediary.

Multi-sig Extension

When any of the AIDS in the protocols above are thresholded multi-sig it means the set of controlling key-pairs for the AID is more than one. Then the basic protocol templates need to be modified. For signing this modification is that at least one signature from the set of controlling keys must be attached or the message is dropped. In order to indicate which key was used to generate any signature, an indexed signature is used. This is a special encoding that embeds the index in the derivation code for the signature. The edge may then collect in escrow single-signed messages, where each is signed by at least one of the keys until the threshold is reached or drop the message if escrow times out before the threshold is reached. Any intermediary may do the collection and escrow before passing along the message. This enables the intermediary to load balance and reduces the memory and processing requirements of the edge.

For encryption, each key in a multi-sig provides a unique encryption destination. This means that for each of the message examples where there is an encryption destination with a decryption key, the ciphertext must appear as a list of ciphertexts each with a destination that corresponds to one of the multi-sig destination AIDs controlling keys. This may require two destinations, one that is the AID in order to look up the key state and the second an index into the key list in order to look up the specific key used to decrypt. Using the index is a much more compact approach and separately listing each destination key.

Interaction Non-Content Metadata

iSAIDs

Other non-content metadata, such as interaction identifiers that “glue” messages together into a conversation, thread, topic, transaction, or interaction, may also provide correlatability across messages. We use the generic term *interaction identifier* (IID). We can think of interaction-specific metadata as the glue that forms an *end-wise* interaction context that is independent of any associated end-wise routing context at the AID level. The end-wise routing context does not need any glue to bind multiple messages together. Each message is independently routable between the AIDs at its ends (edges).

For example, as suggested above in both the Header section and the Relationship Formation Protocol section, the SAID of the first message in an interaction can be used as the interaction ID. This can then be labeled as an *interaction SAID* or iSAID for short. A SAID is a Self-Addressing IDentifier generated from a cryptographic digest of the message. Therefore a SAID is also a UUID, and if the message includes a salty nonce, then the content of the message is not discoverable merely from its SAID. The SAID can be employed as an interaction glue identifier (iSAID) that is cryptographically universally unique to a given interaction. Then another message can insert that iSAID as content into itself, thereby gluing or chaining the two messages together. This allows the two messages to be glued together without repeating any of the AIDs as non-content metadata from the first message (unless they are needed for the authenticity or confidentiality of the new message specifically but not in order to glue to the first message). Any subsequent message in the conversation can use the SAID of the previous message to construct a chained (glued) together conversation, and so on. The set of messages so chained together form an interaction that is a verifiable data structure. Given that each message is signed by its source, then the whole interaction becomes a verifiable data structure that is strongly authenticatable.

The important insight here is that given the same information-theoretic-secure non-correlatability of SAIDs with salty-nonces, the same partitioning principle used for contextually isolating identifiers (AIDs) in a relationship to the context in which that relationship applies can be used for isolating interaction glue identifiers (ISAIDs) to the interaction context in which they apply.

Privacy by Isolating Interaction Contexts

The interaction context can be completely confidential with respect to any communication context that conveys all or part of the interaction messages. This includes an interaction that is split across multiple communication contexts. Properly constructed, there is no mutual correlatability of any of the messages in the interaction except at or by the endpoints, regardless of any intermediaries or any communication context identifiers used to convey the confidential interaction. The communication messages are un-connected and un-glued (except at the level of the communication context for reliability). The embedded interaction header is completely confidential and un-correlatable to any 3rd party viewing the identifiers in the communication context messages.

Thus, the hard problem of protecting against the correlatability of interaction metadata is providing a confidential context in which to embed the interaction metadata. Which we already established above. So given that, we can build confidential and private interactions (“conversations”) by using confidential interaction metadata embedded in confidential data conveyed by communication relationship contexts.

Indeed privacy comes from the fact that as far as any 3rd party is concerned, the communication context relationship is just conveying plain vanilla confidential data in un-connected messages that are unaffiliated with any interaction.

To elaborate, interaction context partitions can be a proper subset of any relationship context partition. A given relationship context partition may have multiple interaction contexts (i.e., multiple conversations). Therefore if the relationship context is private then any of its interactions contexts are also private (QED).

In order to show a practical example we define some new syntax. Let *iAB* denote some pair-wise end-wise interaction context between controllers *A* and *B*. We index specific interactions with a subscript and denote the associated interaction context identifiers as follows: i^0AB, i^1AB, i^2AB. We assume that these identifiers are SAIDs or iSAIDs. An important nuance is that the actual interaction identifiers (iSAIDs) come in two forms. The simplest is to repeat the same identifier in each and every message in the interaction. The second is to chain identifiers where any given identifier only appears in two messages. The message that the iSAID is the SAID of, and the next subsequent message where the iSAID is a reference to the previous message. The sequence of chained SAIDs forms a chaining set that constitutes the iSAID of the interaction. For the sake of simplicity, we will use the former form in the examples below.

Suppose, for example, we start with the non-shared intermediary protocol above. We can make any given message a member of an interaction by including the interaction's iSAID. In the example below, it's iAB_0.

|src ip A_2*, dst ip* C_0*|<[src* A_2*, dst* C_0*, {src* A_2*, dst* D_2*, dst* D_0*, <[src* A_1*, dst* B_1*,* i^0AB*, {src* A_1*, data}*B_1*]>*A_1*}*C_0*]>*A_2

In the example above, i^0AB appears as plaintext at the same level as the AIDs *src* A_1, and *dst* B_1. All three of these are confidential with respect to the hop-wise communication contexts. No 3rd party can correlate the messages in the interaction.

Recall that the intermediaries are trusted 2nd parties. As trusted 2nd parties, the intermediaries can correlate the end-wise routing AIDs *src* A_1, and *dst* B_1. But now, given the presence of iSAIDS, they can also correlate subsets of those end-wise routed messages at the AID level as belonging to distinct interactions. In some applications, *A* and *B* may wish to make the interactions private with respect to the intermediaries. They can do this by putting the iSAIDs inside the confidential content of the end-wise context messages as follows:

|src ip A_2*, dst ip* C_0*|<[src* A_2*, dst* C_0*, {src* A_2*, dst* D_2*, dst* D_0*, <[src* A_1*, dst* B_1*, {src* A_1*,* i^0AB*, data}*B_1*]>*A_1*}*C_0*]>*A_2

Now only *A* and *B* can correlate messages that belong to the interaction identified by i^0AB. This has effectively created a new partitioned interaction context for i^0AB that is confidential with respect to the end-wise routing context given by the relationship, *(*A_1*, <>,* B_1*)*. As far as the intermediaries are concerned, all messages in the end-wise routing context, *(*A_1*, <>,* B_1*)* are undifferentiated from each other, thereby providing herd privacy to the interaction context *isup>0AB<*.

Privacy comes from the fact that messages in the end-wise routing context look like plain vanilla messages as far as any interactions are concerned.

Cross Relationship Context Interaction Contexts

A complication arises when one wants to have an end context span multiple relationship contexts. Typically for communications purposes, when there are multiple communications media or intermediaries. But, given that one can form a private relationship by reference using the relationship formation protocol defined above, then

one can avoid having any transaction spanning multiple relationships by first forming a relationship for such a transaction. And then leveraging those pre-existing communication contexts to confidentially (and privately) embed that relationship and its associated transaction (conversation) with its transaction metadata.

Suppose *A* and *B* wish to use more than one end-wise routing context for a given interaction. This generalizes to any number of hop-wise communication contexts, but for the sake of simplicity, we will just use the one established above. Let *A* and *B* establish another end-wise relationship denoted by $(A_3, <>, B_3)$. This could have been formed using an OOB setup or using the relationship formation protocol leveraging the existing relationship $(A_1, <>, B_1)$.

Let the interaction be denoted i^0AB. There are two different ways we can leverage the new end-wise relationship.

The first way is to use it as yet another end-wise routing relationship at the AID level. A message using this approach is shown below:

$|src\ ip\ A_2,\ dst\ ip\ C_0|<[src\ A_2,\ dst\ C_0,\ \{src\ A_2,\ dst\ D_2,\ dst\ D_0,\ <[src\ A_3,\ dst\ B_3,\ \{src\ A_3,\ i^0AB,\ data\}B_3]>A_3\}C_0]>A_2$

Any message in that same interaction can be conveyed with either of the end-wise routing relationships. This would be most useful if each of the end-wise routing relationships were one-way. In that case, messages from *A* to *B* would use a different end-wise relationship than messages from *B* to *A*. Any 3rd party, as well as any intermediary, could not correlate messages that went from *A* to *B* with messages in that same interaction that went from *B* to *A*.

The second way to use this new end-wise relationship is to embed it in the pre-existing end-wise relationship. This makes the existence of this new end-wise relationship $(A_3, <>, B_3)$ hidden from not just 3rd parties but also the 2nd party trusted intermediaries. A message using this approach is shown below.

$|src\ ip\ A_2,\ dst\ ip\ C_0|<[src\ A_2,\ dst\ C_0,\ \{src\ A_2,\ dst\ D_2,\ dst\ D_0,\ <[src\ A_1,\ dst\ B_1,\ \{src\ A_1,\ <[src\ A_3,\ dst\ B_3,\ i^0AB,\ data]>A_3\}B_1]>A_1\}C_0]>A_2$

The embedded relationship's interaction context can be hidden from the routing relationship's context by making that identifier and data confidential as follows:

$|src\ ip\ A_2,\ dst\ ip\ C_0|<[src\ A_2,\ dst\ C_0,\ \{src\ A_2,\ dst\ D_2,\ dst\ D_0,\ <[src\ A_1,\ dst\ B_1,\ \{src\ A_1,\ <[src\ A_3,\ dst\ B_3,\ \{src\ A_3,\ i^0AB,\ data\}B_3]>A_3\}B_1]>A_1\}C_0]>A_2$

This approach can be used with any hop-wise communication context and any end-wise routing context between *A* and *B* to convey an interaction between *A* and *B* that is strongly authentic, confidential, and private with respect to all 3rd parties and any 2nd party trusted intermediaries.

Combined Source Vector and Table Routing

The non-shared intermediary protocols above use a combination of both source vector routing and table routing. This splits and isolation the routing information so that third parties cannot view a full route without compromising at least two parties (ends or intermediaries) and any near-end or near-side intermediary cannot view a full route without compromising at least either the far-side intermediary or far-end. Splitting the routing

path into two components means no single source of failure exists. This means that only part of the route is transmitted from the near-side source. The remainder of the route must be filled in from the far-side routing table. The source vector part is only ever in memory so that compromise of the routing table in persistent storage does not reveal the full route. Effectively the far-side intermediary acts like a dead drop controlled by the far-side destination to which the near-side source drops messages. The near-side source never sees the far-side eventual destination, just the far-side intermediary as a dead drop.

Sustainable Privacy, Authenticity, and Confidentiality

Time Value of Information

In general, privacy dissipates over time. This is because digital information is inherently leaky, and those leaks become more correlatable as the body of leaks grows over time. This leakiness can be balanced by the diminishing exploitable time value of correlated information.

In general, the primary time value of correlated information for data aggregators is that it can be used to predict behavior. Advertisers want to predict who will most likely be receptive to their marketing campaigns. The predictive accuracy of aggregated behavioral information of potential participants in any given market is largely a function of the nearness in time of the behavior used to make the prediction. We can ascribe a time constant to a given market's exploitable predictive potential where information older than the time constant no longer has net predictive value in excess of the cost of aggregating it. Information that exceeds this time constant is considered stale because there is no longer any incentive to aggregate and correlate it. Therefore, in spite of the fact that privacy dissipates over time, the value of correlation also diminishes over time so that cost-effective privacy protection mechanisms can focus resources on near-term correlatability. This provides a sweet spot for sustainable privacy protection that is governed by the time constant of the time value of exploitable correlation. Likewise, the cost of privacy protection can be weighed against the cost due to the harm of exploitation. If the cost of protection exceeds the cost due to the harm of exploitation, then it's not worth protecting (i.e., it's counterproductive to the protector). If the cost of correlation in order to exploit exceeds the time value return of exploitation, then it's not worth exploiting (i.e., it's counterproductive to the exploiter).

The protocols defined above provide mechanisms for granular partitioning contexts so that correlatability is also granularly partitioned. This enables one to control the exploitable time value of the correlatable information that can be leaked out of that context. Once a context has become leaky, a new isolated context can be created that restarts the clock on time value correlatability. This provides a trade-space between the friction and cost of forming and maintaining contexts, the length of time before a given context becomes leaky, and and the time constant of the exploitable value of leaked information. The cost of protection includes expensive one-time OOBA setups. If the leakiness of a given context is cost-effectively protectable beyond the time constant on the time value of exploitation, then new information is sustainably protectable indefinitely.

Sustainable Management of Contexts

A party wishing to protect against exploitation via correlatable metadata (identifiers) sustainably has a similar problem to that of the operatives in covert and/or clandestine operations.

Covert
A covert operation is an operation that is planned and executed in secrecy so that the identity of the agency or organization remains unknown or is plausibly deniable.

Clandestine
A clandestine operation is an operation that is carried out in such a manner that the operation remains in secrecy or is concealed.

(see DoD Dictionary, Covert vs. Clandestine)

From the standpoint of a covert operative, the fact that the organization conducting the operation remains unknown puts the operative at great risk. The organization can burn or disavow the operative without incurring liability to the organization itself. This means the operative must have a legitimate public cover identity. Whereas a clandestine operative has no such public cover, should the operative be discovered then that discovery would incur liability to the orgainzation itself. This means clandestine operations have more limited use cases. Often constrained but the technical and operative challenges of maintaining the privacy of the whole operation, including all operatives. Consequently, covert operations have more use cases that come at the cost of setting up public cover identities for the operatives.

As a result, A covert operative must be able to survive long-term without being found out. This means they must have a public cover identity that masks the private behavior they wish to hide. Effectively, their public cover identity allows them to hide other private behavior in plain sight. In comparison, a clandestine operative has no such public cover but must operate totally privately. Any leakage at all by a clandestine operative of its private behavior becomes a strong signal that can be correlated to detect their operation. As a result, a given clandestine operation is unsustainable long-term. They tend to be short-term, in and out. In contrast, covert operations can be much more long-term because the leakiness of private behavior is masked by their public cover behaviors.

The analogy for our purposes is that many approaches to internet privacy have tended to look like clandestine operations where the whole operation must be concealed and that concealment is largely technological. But any leakage at all jeopardizes the whole operation. This makes the approach fragile and largely unsustainable. In comparison, a more covert operation approach may be more robust to leakage and hence more sustainable. For example, there are various ways to provide the equivalent of a “cover” identity for communications traffic. Specifically, with regard to the protocols above, parties *A* and **B** have several relationship contexts. They each have a hop-wise communications context with an intermediary, e.g. *A* with *C* and *B* with *D*. To the extent that *C* and *D* have many users that have well-known legitimate “public” uses of *C* and *D*, then *C* and *D* each have a cover identity with respect to their respective relationships with *A* and *B*. For example, if the only time anyone uses *C* is to conduct some private activity, then the use of *C* by itself is suspect and provides correlatable information. Specifically, suppose that *C* is a VPN, and the only time *A* uses *C* is to communicate with a crypto-currency exchange in Cyprus in order to avoid the FATF KYC rules for exchanges in FATF-compliant countries then, anytime *A* uses *C* the use is exploitable information. Whereas if *A* uses *C* for all communications, then the occasional communication to Cyprus is concealed by the cover of the other activity. Likewise, if all users of *C* use *C* for all their traffic, then they provide herd privacy to every other user. But if most of the users of *C* only use *C* to conceal transactions for which they want more privacy (i.e., clandestine), they make the mere use of *C* itself a correlatable signal that exposes their clandestine operation.

The nesting of contexts extends the covert analogy. Given *A* has a generic end-wise routing relationship context with *B*, that *A* uses for a multitude of confidential interactions with *B* where each interaction uses a dedicated interaction relationship context between *A* and *B.*, then the end-wise routing context provides cover for the end-wise interaction contexts. The traffic over the general routing context is another form of herd privacy. This enables *A and* B* to conduct sustainable hide-in-plain-sight operations that minimize the exploitable correlatable information. Each of the relatively public long-term generic communication contexts (covert) and long-term generic routing contexts (covert) provide nested covers to the really valuable private short-term interaction contexts (clandestine).

Relationship Discovery Protocols

The protocols above assume that at least one relationship between any given pair of controllers given by *A* and *B* has been formed in a secure way which includes wich least one OOBA factor. We call this an OOB setup. We have not covered in any detail what various mechanisms may be employed to set up the initial relationships. We did, however, notionally outline a relationship formation protocol that can be used to form other relationships in a secure way, given the pre-existence of the first relationship. Essentially relationship formation is mostly about the discovery of the AIDs used for that relationship. That discovery can happen OOB in an insecure way but then be verified in-band (IB) before any sensitive data is exchanged. Or the discovery can happen in an authentic, confidential, and private manner.

With KERI, we don't exchange public keys per se but AIDs that each has a verifiable key public key state. One nuance is that a given cryptographic public key or AID may be actually meant to be held confidential between two parties. Making it not so public. Likewise, a given relationship consisting of a pair of AIDs may want to be confidential and/or private with respect to 3rd parties. Therefore different AID discovery mechanisms (setups) might themselves involve more complex interactions.

This seems to be the area where some of the patterns in DIDComm would benefit from being recast from an AID discovery or setup perspective. The DIDComm protocol includes different patterns for exchanging public keys as well as DIDs. The latter (a DID) is similar to an AID, although without the same security properties with respect to its key state. Indeed, a did:keri DID provides a DID-compatible version of a KERI AID but with the security properties of KERI preserved.

Finally, KERI has associated AID discovery protocol called the OOBIprotocol (out-of-band-introduction). This provides a bare-bones approach that covers many common use cases. This may be extended with other protocols for other application use cases. This may be one of the most fruitful ways to leverage the work done on DIDcomm.

CESR

One of the important principles is that we can leverage the information-theoretic secure properties of cryptographic primitives as identifiers for the purposes of privacy. The main reason for CESR was to make the use of cryptographic primitives sufficiently convenient that they could be used in protocols in ways that could not have been ever contemplated before given the much more verbose and inconvenient representations such as JWTs.

An agile cryptographic primitive can't be minified like a JSON label or Javascript variable name; otherwise, it loses its entropy and its agility. But we can represent it in the most compact possible form as a string that is composable to binary (which CESR does uniquely).

Appendix DDOS

DDOS Protection Efficiency

DDOS protection mechanisms can be compared using what may be called a DDOS protection efficiency metric. In general, DDOS protection efficiency is increased by being able to identify as far upstream as possible any information about a given packet or connection that allows the protection mechanism to drop the packet. The further downstream the packet goes before that decision can be made, the less efficient the DDOS protection mechanism. For example, if a DDOS protection mechanism has to first decrypt a packet in order to verify a signature on that same packet before it can make the decision to drop the packet, and given that decryption and signature verification have roughly the same processing load, then the DDOS protection efficiency is half what it would be for a packet where the decision to drop only requires a signature verification. If the protocol has nested (encryption signature, encryption signature ...) layers that must all be processed before the decision to drop may be made then the DDOS protection efficiency is multiplicatively reduced by the addition of each nested layer. This is true no matter where those nestings occur in the processing chain. It doesn't matter whether the nested processing occurs only at the final processor or is divided across a series of upstream processors. The DDOS efficiency of the overall system is exponentially reduced by each nested layer. So multiplying nesting layers directly impacts DDOS efficiency. This is a harsh trade-off that should be considered when designing protocols.

Drop decisions that require intimate knowledge of identifiers besides the IP packet headers usually have significantly reduced protection efficiency. This is because the first line of DDOS defense is firmware load balancers that can make drop decisions merely on the IP source in combination with the protocol type. The next line of defense is software load balancers like nginx that can make more complicated drop decisions by looking into the packet. But the protection efficiency of a software load balancer may be 1/10 or less than the protection efficiency of a firmware load balancer. The third line of defense is the edge host itself running on a server. This host may be able to understand the packet in more detail. But the protection efficiency of the edge host may be 1/10 or less than that of a software load balancer. As a result, any protocol that can only make a drop decision at the edge host may have a protection efficiency that is 1/100 of one that can make a drop decision at a firmware load balancer. Any protocol that requires decryption in order to make a drop decision can only make that drop decision at the edge host because only the edge host has the decryption key. Sharing those keys with upstream load balancers breaks confidentiality and exposes the protocol too much worse attacks than mere DDOS. But drop decisions that require a signature verification given a public key can be performed upstream of an edge host (in a software load balancer like nginx) with a corresponding multiplicative increase in protection efficiency.

This same analysis applies in general to overall system DDOS efficiency. If any intermediary can make a drop decision because it has a whitelist of other intermediaries from which it will accept packets then it only has to process a signature verification. If, instead, the intermediary has to first decrypt before it can use its whitelist to verify it already has 1/2 the DDOS efficiency, and if each of N intermediaries in the path of the packet has 1/2

the efficiency for the same reason, then in comparison, the DDOS efficiency is 1/2N versus a protocol that can drop based on a signature without decrypting first. So if no decrypt must be done in order to drop (as in at end-only-viewability) then we can have very efficient DDOS protection.

This does not mean that protocols can't have nested encryptions. The DDOS protection efficiency is the ratio of malicious packets that a DDOS adversary can throw at a given endpoint that can be dropped without degrading the endpoint vs. the number of honest packets that the endpoint can sustain absent a DDOS attack. So if an endpoint can sustain X honest packets per second and can also drop N*X packets per second then the DDOS protection efficiency is N times the sustained throughput.

Security-by-obscurity

In protocol design for security with respect to DDOS protection, we also have to be careful not to make the mistake of security-by-obscurity. Any endpoint of value will eventually be discovered because its value provides incentives to search for that endpoint. The mere fact that a potential DDOSer does not easily know up front what services to DDOS attack has nothing to do with DDOS protection efficiency or DDOS vulnerability. It has to do with DDOS susceptibility, which protection is at best ephemeral and analogous to security-by-obscurity.

For example, *mix* networks by design make it difficult for a load balancer to correlate malicious packets to a given IP source. So interposing mixers as intermediaries usually significantly reduces the DDOS protection efficiently of the edges of that network. Mix networks are also vulnerable to what is called a sleeper attack, where a malicious attacker can correlate across mixers and discover the eventual endpoint. Once they have performed this correlation, the adversary can DDOS the endpoint directly, thereby avoiding any load-balancing protection provided by the mixer. Unless of course, the mixer has a whitelist that allows it to drop any un-permissioned injection of traffic by a sleeper attacker.

Amplification Attacks

Of particular concern are protocols that have an amplification attack vulnerability. Any exponential reduction in DDOS efficiency in any protocol stack becomes a juicy target for an amplification attack. The attacker constructs malicious packets that amplify the resources needed to make the drop decision. Amplification attack vulnerabilities are a sign of a poorly designed protocol. The less processing that must be done in order to make a drop decision the better. Any processing that is amplified by maliciously crafted packets is disproportionately vulnerable to DDOS attacks.

For example, A DDOS amplification attack vulnerability makes a W3C VC based on JSON-LD generated RDF with an embedded RDF URDNA–2015 integrity-proof problematic, especially at scale. One can easily construct a malicious JSON-LD VC that triggers an NP-hard amplification of the time to verify the JSON-LD signature integrity proof (when using URDNA–2015) on the receiver's end. This is because RDF blank nodes can induce recursive loops in URDNA–2015 that are of order (NP) hard. It's trivial to construct a malicious JSON-LD document that induces these loops. The receiver can't make a drop decision until after it has transformed the JSON-LD into an RDF graph and then computed the integrity proof on that graph. Any malformed packet that exploits any extra processing not merely in the RDF integrity proof but the RDF expansion including any @context or @vocab substitutions makes the receiver disproportionately vulnerable to DDOS amplification

attacks. A mitigation would be to wrap any JSON-LD -based VCs in an external proof (such as an ACDC container) of the over-the-wire packet with a drop decision based on verifying the external proof (container) such as a simple signature against a white list of approved senders. The packet gets dropped before the container is opened.

One of the business models for DDOS attackers is DDOS ransomware. According to CloudFlare, 16% of DDoS attacks are ransomware. Thus an attacker needs only mount a short-term attack to prove a vulnerability that a more intensive attack could exploit in order to monetize via ransomware.

Appendix: Using Crypto in a way that is Too-Clever-By-Half

too clever by half:
idiomatic, of a person, plan, theory, etc. Shrewd but flawed by overthinking or excessive complexity, with a resulting tendency to be unreliable or unsuccessful.

One of the repeated mistakes that protocol designers make is in thinking that they can mix and match cryptographic constructs like encryption and signatures in clever ways in order to solve all of the different variations of higher-level communication tasks. The ESSR framework basically tells us that there is really only one secure approach to mixing encryption with signatures. Any other approach exposes vulnerabilities that either have been or will be exploited given sufficient incentive and time.

Higher-level communications tasks may require repeated application of the ESSR construction in combination with OOB setups and intermediaries in order to achieve the desired goals. The combined effect of multiple out-of-band setups with in-band mechanisms may be required to provide the richness essential to some tasks without exposing them to vulnerabilities that make the protocol useless once attacked. Unfortunately, these combined schemes may become too complex with too much friction to be widely adoptable so it may be a problematic trade-off. There may not be a viable solution in general for all desired communication tasks, especially for privacy protection against state actors who have too many attack vectors vis-a-vis practically adoptable protection mechanisms.

Appendix: 1st Party Non-Viewability

One reason to use PKE is so that the encryptor (1st party) can delete the plaintext from their system and avoid any liability for continued storage of the plaintext. Thus the data is no longer viewable by the 1st party. This would be applicable for GDPR data processors, in order that they may retain the encrypted-to-the-destination ciphertext as non-viewable by them but held for the purposes of the end-destination. Or the only time the plaintext was viewable was in memory, never on disk. This makes for more zero-trust computing. Anyone auditing the 1st party code can verify that the first party is deleting the plaintext and never storing it after that. Moreover, any intermediary that had to transform and then re-encrypt can employ this property.

But end-only-viewability is also reflexive. any party can decide that they themselves are the other end and encrypt it to themselves thereby enabling them to delete the plaintext. This is the zero-trust principle of encrypted-at-rest, not merely encrypted-in-motion. This is important when the 1st Party is distributed across multiple infrastructures. This means the private decryption key can be isolated to only one part of the 1st

party's infrastructure, and the other parts can encrypt to that isolated part such that only the isolated part can decrypt. Using symmetric encryption, on the other hand, would require exposing the shared secret encryption key with all the parts thereby weakening it through increased exposure.

To restate, given a distributed 1st party where that 1st party doesn't want any plaintext hanging around, it can enforce on itself a policy of end-part-only viewability by using PKE.

But yes, it's confidentiality via non-viewable ciphertext vs. viewable plaintext. The confidentiality mechanism is that once a 1st party decides to make the plaintext end-only viewable, they can enforce that by deleting their plaintext copy. When the 1st party decides to encrypt the plaintext to themselves, they can then delete their copy of plaintext but be able to view it later. In this case, there are two different ends that have end-only viewability of the same plaintext.

The point is that it gives any source the tools to enforce which ends have end-only viewability, including itself.