

FDC3

Identity and Security 2025

DRAFT PROPOSAL BEING WORKED ON BY

FINOS' FDC3 Identity & Security Working Group

Contents

[Contents](#)

[Executive Summary](#)

[Support](#)

[Introduction](#)

[Problem Statement](#)

[Use Cases](#)

[Solution](#)

[Benefits](#)

[Conclusion](#)

[Appendix A: Security Issues Being Addressed](#)

[Appendix B: New Standard Mechanisms](#)

[Appendix C: A FINOS Circle of Trust](#)

[Appendix D: Some Criticisms / Drawbacks](#)

Executive Summary



The Financial Desktop Connectivity and Collaboration Consortium (FDC3) has long facilitated interoperability among financial applications through client-side protocols, primarily within secure desktop containers.

However, as the ecosystem evolves toward **web-based implementations**, new security challenges arise in an open, decentralized environment.

This paper presents a comprehensive security enhancement for FDC3, addressing critical issues such as **application spoofing, data leakage, unauthorized access, and the limitations of bilateral trust relationships**.

It also explores key use-cases that require FDC3 user- and app- identities, such as requesting privileged information, working across devices, sending messages on a user's behalf and simplifying the log-in process across heterogeneous applications as well as potentially expanding the scope to cover agent-based activities enabled by AI.

By integrating robust cryptographic mechanisms—including **digital signatures, encrypted channels, and JWT-based user identity**—and introducing the concept of **Circles of Trust**, the proposed framework establishes verifiable app and user identities while enabling scalable, secure data exchanges.

Support

Many firms are supporting the FDC3 Identity & Security initiative. Here's a selection of quotes.

 <p>Paul Goldsmith Executive Director</p> <p>Morgan Stanley</p> <p>"Morgan Stanley is deepening its commitment to FDC3 by collaborating with peers at other financial institutions on the FDC3 Security and Identity strategic initiative at FINOS."</p> <p>"This initiative builds on the foundational work we demonstrated at OSFF in October 2024 in our cross-firm FDC3 demonstration with Wellington Management. We're looking forward to seeing security and identity primitives added to the FDC3 standard to allow for deeper, more coordinated, cross-industry workflows of this type"</p>	<p>"Trust and identity are critical elements for seamless inter application communications on the Desktop. We are excited to see how this enhancement to the FDC3 standards will further reduce barriers especially around communicating with third party applications."</p>   <p>Bhupesh Vora Managing Director</p>
 <p>Dietmar Fauser CIO</p> <p>SYMPHONY</p> <p>"Symphony has grown to become a communication backbone for the financial services industry by being above all a network of certified identities. Whether those identities are users, teams, bots, apps - this is the layer of trust on which all our users' communication and collaboration workflows are built. So obviously we were vocal proponents of introducing the concepts of Security and Identity to FDC3."</p> <p>Our Senior Engineering Director, Yannick Malins, leads the FDC3 Security and Identity workstream, and we're excited to see firms across the financial services sector benefit from the secure workflows and more tightly integrated application suites this enhancement to the standard is set to deliver!"</p>	<p>"NatWest is a large organisation and therefore has a complex FDC3 estate: multiple desktop agent vendors and both internal- and external- facing deployments. This advance in the standard really speaks to our pain points."</p> <p>"By delivering standard FDC3 constructs for identity and security primitives, it will remove one large hurdle faced by our app developers in getting their software into the hands of clients faster and more securely."</p>   <p>Matthew Harvey Head of Cross Product Sales and Development</p>

Introduction

About FDC3

The Financial Desktop Connectivity and Collaboration Consortium (FDC3) is an open standard designed to enable seamless interoperability between financial applications. Created under FINOS (the Fintech Open Source Foundation), FDC3 defines common protocols, APIs, and data formats that allow applications to communicate securely and efficiently across different platforms.

Unlike traditional server-side integration approaches, FDC3 operates as a **client-side protocol**, similar to how HTML standardizes content rendering in web browsers. This means applications running on a user's desktop can interact in real time without requiring centralized infrastructure, reducing integration complexity and fostering a more connected financial technology ecosystem.

Historically, FDC3 has been closely associated with **desktop containers**—specialized environments that enable financial applications to communicate within a controlled ecosystem. However, as web technologies continue to evolve, there is a growing shift toward **FDC3 on the web**, where interoperability extends beyond containerized desktops to browser-based applications.

FDC3 On The Web

To interop on the web, previous versions of FDC3 relied heavily on **custom vendor libraries** being added to apps. While effective, this approach often led to fragmentation and vendor lock-in, limiting the flexibility of financial institutions. **FDC3 2.2 removes this dependency by introducing native web support**, allowing applications running in standard web browsers to communicate using the same FDC3 protocols as their desktop counterparts. This means financial firms can now integrate and orchestrate workflows across both **containerized and web-based applications** without requiring proprietary infrastructure, opening the door for broader adoption across the industry.

While **FDC3 2.2** enables greater flexibility by extending interoperability to the web, it also introduces new security challenges. Unlike desktop containers, which operate within controlled environments, **the web is a far more open and hostile computing landscape**, where applications run in diverse, often untrusted contexts. This shift raises critical questions about **identity, authentication, and authorization**—who is making an FDC3 call, whether they are authorized to do so, and how to prevent unauthorized access or data leakage. Without a robust security model, web-based FDC3 implementations risk exposure to **cross-origin threats, session hijacking, and malicious actors impersonating trusted applications**. As FDC3 expands beyond closed ecosystems, implementing strong identity verification and security controls becomes essential to ensure safe and compliant interoperability.

But opening up FDC3 to the web **introduces new security challenges**: application spoofing, cross-origin attacks, data leakage concerns, session hijacking and trust management issues (see Appendix A for a more in-depth discussion of these).

Prior Art

Wellington and Morgan Stanley's 2023 FDC3 Demo

Wellington Management and Morgan Stanley, both leaders in the financial industry, conducted a **demo using FDC3 in 2023** to showcase interoperability across financial applications between firms. Their goal was to share **liquidity data** — essentially, financial holdings information — across different systems.

However, despite the promise of seamless integration through FDC3, both firms encountered significant issues: they had no way to verify the identity of their partner and there were risks to data integrity from sharing sensitive information over FDC3 - a compliance / regulatory concern. See Appendix B for a more detailed breakdown.

Symphony's 2023 Demo: Sharing a JWT Token Over FDC3

In 2023, **Symphony**, a leading provider of secure communication and collaboration tools for the financial industry, [demonstrated how to share a user's JWT token over FDC3](#). JWT tokens are widely used in web applications for **authentication** and **authorization**—essentially, they prove that the user is who they claim to be and define their access rights within an application.

However, this demo *again* brought to light new challenges: token data leakage, for one, passing around tokens in public via FDC3 and whether or not apps could trust the tokens they received from each other. (See Appendix A for a more in-depth discussion of these).

Wellington and Morgan Stanley's 2024 FDC3 Demo

In 2024, these firms returned to OSFF and performed the [same demo as the previous year](#), however this time demonstrating two key advances:

1. **Signed FDC3 Context Objects** which meant that applications receiving them could guarantee the origin and integrity of the data.
2. **Encrypted FDC3 Private Channels** which meant that position information could be sent in secret between the two firms over the ordinarily open protocol of FDC3.

This demo demonstrated these features **layered over the top of unmodified FDC3 desktop agents** suggesting an easy upgrade path for apps interested in these privacies. While this

mitigates the problems with the 2023 demo, it introduces **new concerns, addressed by this white paper**:

1. Sensitive Data Leakage: Can Apps Be Trusted with Data?

While digital signatures ensure data comes from a trusted source, they do not guarantee that the receiving app has the appropriate **security controls** in place. This introduces the possibility that an app might:

- Store sensitive data insecurely.
- Share data beyond its intended recipient.
- Use the data inappropriately or for malicious purposes (for example, insider trading or corporate espionage).

2. App Identity vs. User Identity

The digital signature proves that the app sending the context object is authenticated and has not tampered with the data, but it does not authenticate **which user** is behind the action (as was the case in Symphony's demo).

3. Lack of a Standard

There was no clear standard for implementing **signed context objects** within the FDC3 framework. Institutions and developers had to determine how to handle key management, signature verification, and integration with existing authentication and authorization systems on their own.

4. Pre-Existing Bilateral Trust Relationships:

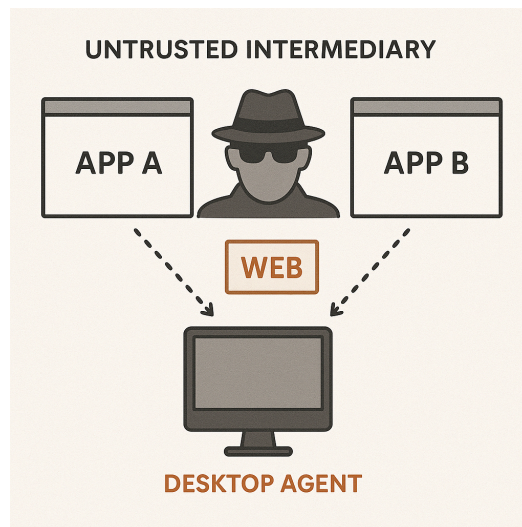
In the demo, apps required bilateral trust relationships—i.e., the apps involved had to know about each other and explicitly trust one another (via use of certificates and keys). This limited the scalability and flexibility of the solution, as it was not easily applicable to broader, decentralized ecosystems of apps.

Desktop Agent Trust and Vendor Approaches

In traditional desktop environments, a dedicated desktop agent acts as a trusted intermediary to manage FDC3 communications, ensuring that only approved applications exchange data under controlled conditions. This agent is often pre-configured by the host system, so its trustworthiness is implicitly established through its integration within a secure desktop ecosystem.

It is therefore possible that some of the problems above can be solved by desktop agent vendors. For example, if a desktop agent is trusted, it could be used to filter messages between apps to enforce data leakage policies. It could also manage app identity, or perform OAuth-style user login functions on behalf of apps, centralising login.

However, replicating that level of trust on the web is inherently more challenging due to the open and decentralized nature of browser environments. On the web, there isn't a single, universally trusted desktop agent. In fact, as discussed above, **FDC3 For The Web is trying to move away from specific vendor libraries.**



It's not therefore desirable that applications are forced to rely on predefined lists of trusted endpoints or agents to verify the integrity of inter-app communications. Desktop Agents (much like browsers) should therefore be regarded as an untrusted environment.

Problem Statement

FDC3 Security aims to address 6 key problems, as shown in the diagram below and the accompanying text breakdown.



1. Shift to Web and Increased Security Needs

As FDC3 moves to the web, the need for enhanced **security** and **identity management** has become critical, particularly as financial applications interact in increasingly **hostile computing environments**.

2. App Identity Limitations

The current FDC3 protocol doesn't assert app identity and lacks mechanisms for ensuring the **trustworthiness** of sensitive data being exchanged, exposing it to risks of **data leakage** and **manipulation**.

3. Insufficient User Authentication and Authorization

Outside of the Symphony demo, **User authentication** and **authorization** are not adequately addressed, creating potential for **unauthorized access** and **insider threats** when data is shared between users within trusted apps. When apps are left to handle their own logins, this leads to a profusion of login approaches for users (OAuth, challenge/response, access keys on a per-app basis).

4. Lack of Data Integrity Assurance

Existing solutions do not provide guarantees of **data integrity**, leaving financial institutions vulnerable to errors, fraud, and breaches during inter-app communication.

5. Pre-Existing Bilateral Trust Relationships

In the **2024 Wellington and Morgan Stanley demo**, apps required **bilateral trust relationships**—meaning the apps involved had to explicitly know and trust each other. This limited the scalability of the solution and made it difficult to apply in a broader ecosystem of apps, where automatic trust and interoperability across different platforms are essential.

6. Need for Evolution in Security Frameworks:

To ensure secure, reliable data exchanges, FDC3 must evolve to address **user-level security** and provide stronger frameworks for **data protection** and **user accountability**, while also enabling **broader interoperability** without requiring app-specific bilateral arrangements.

Use Cases

1. Request Pre-Trade Information (App-Identity)

Persona: Buy-side trader

From: In-house platform

To: Multiple Single-dealer platforms

Flow: A buy-side trader requests margin requirements on an instrument from multiple sell-side platforms. Information is sent back through intent replies. It has a short list of applications it trusts to return this information to.

Risks:

- Who did the request come from?
- Assuming it's a valid request, who else is listening?

2. Sending A Message On A Chat Platform (App-Identity)

Persona: Buy-Side Trader

From: Research Application

To: Another member of staff

Flow: The trader wants to notify a colleague of some new research, and hits a button to send a message on Chat Platform. Chat Platform receives the intent, and observes that the application sending the intent is within a “circle of trust”, and decides to allow the message to be sent.

If the sending app is trusted, you may not need a secondary, on-screen confirmation.

Another example: Creating an open order in a blotter.

Risks / Questions:

- We don't want *any* app to be able to send/spam messages on Chat Platform ad-hoc.
- Who is allowed in the "circle of trust"? *Is this the same for everyone in the firm?*
- Does this mean that administration of this circle is in scope?

3. Auditing / Enabling User Activity (User Identity)

Persona: A Research Application

From: A Trader, using FDC3

Flow: An application wants to keep track of who is reading news articles. Subscriptions are required to view certain articles. The app uses a "circle of trust", leveraging the IDP identity to see who is reading each article.

Risks:

- Who is interested in having this? Vendors?
- Which apps are in the circle of trust? Which apps can request identity from an IDP?
- Does this replace SSO? - Not really.
- *Prospect customers as an example –a legal requirement, knowing who you are talking to.*

4. Multiple Logins (User Identity)

Persona: A Trader

Flow: A trader is starting up his desktop. They need to log into multiple applications, remembering multiple usernames and passwords.

Risks:

- Having multiple usernames creates a security risk as users need to manage more passwords.
- We need to be careful about token spillage, and ensure tokens are scoped to individual apps and are not useful outside of the context.
- "Secondary platforms" - where people aren't bothering to log in because there are too many screens.
- This could take advantage of an existing SSO / Oauth platform if a bank has one.

5. Multi-Party Provenance Example

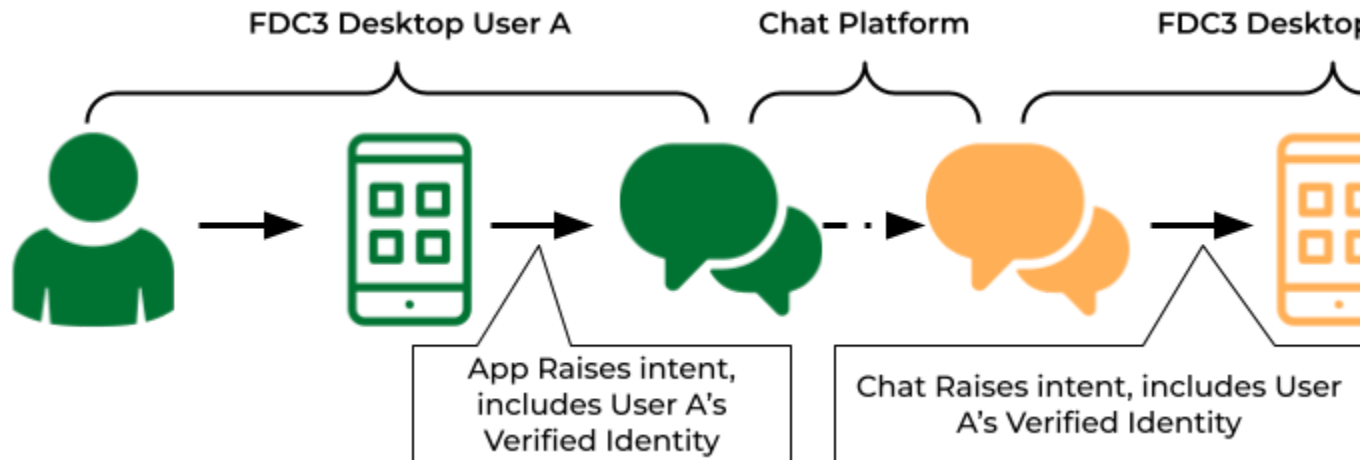
Persona: Buy-side trader

From: In-house platform

To: Multiple Single-dealer platforms And A Chat System

Flow:

- A buy-side trader requests margin requirements on an instrument from multiple sell-side platforms.
- A chat system can be used to send the actions across the network to other users/firms to perform the action.
- The identity of the user is passed with the action.
- At the other end of the chat, an application reads the message and processes it, and is able to raise an intent with the provenance of the original request attached to it.



Implications:

- **App B** would be able to know that it was **User A** that created the request.
- App B doesn't even have to know that it was Symphony/Teams/Other Chat App that sent the message.

Risks:

- From the buy-side perspective, they expect the same level of confidentiality as in cases 1 and 2 - none of their data is shared, but the fact that they want to see this data is in of itself confidential. From the sell-side perspective they cannot risk this information leaking to anyone except the buy who requested it.

Solution

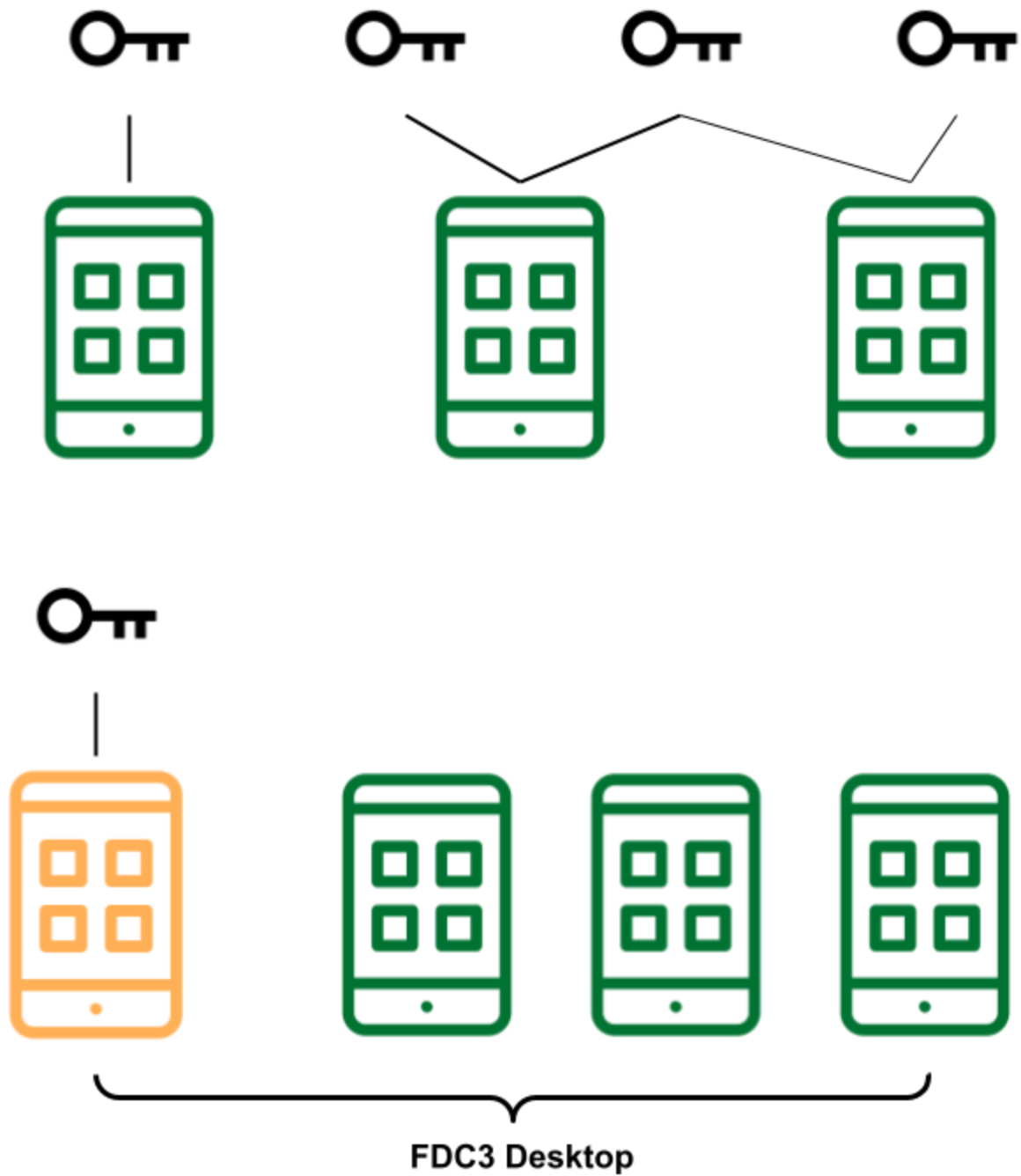
There are various components to the solution to this problem which will be described below, building on the good work that has been described in the introduction section. This includes:

1. **Changes to the FDC3 standard itself** such as new context objects and intents as well as new metadata
2. **New Roles For Applications** as either identity consumers or identity providers (IDPs)
3. **A FINOS Circle of Trust** around applications fulfilling the above roles within the FINOS FDC3 financial ecosystem.

4. **Application Conformance** as a way of gate-keeping entrance requirements for joining the circle of trust.





Why Not Just OAuth / SSO?

1. OAuth has to be decided up-front by application developers. FDC3 allows for this to be done at the time of app-directory construction.
2. Integration for an application therefore becomes a problem, especially for smaller vendors who would end up having to support a whole bunch of different identity providers, many of which might exist behind a firewall and not be accessible to them.
3. Also, being able to pass around identity as part of the workflow is helpful (see use case 5)



Standards Changes

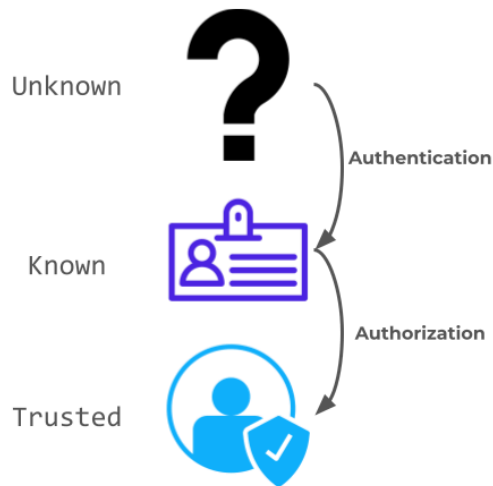
Many of these proposals originate in the <https://github.com/finos-labs/fdc3-security> project, which was the basis of the 2024 Wellington / MS demo.

-  **Proving Data Authenticity:** Apps can now **sign the messages they send**, helping others verify who really sent them and whether the data was tampered with. This relies on securely published public keys and an optional “circle of trust” between apps.
-  **Encrypted Messaging:** Apps in a private group (channel) can **request and share encryption keys** to ensure their communications remain private. Only the intended recipient can read the encrypted message.
-  **User Identity Sharing:** A new data type lets apps securely **share a user's identity** (like name and email), backed by a signed token (JWT) to prove it's authentic and hasn't been faked.
-  **How Identity is Requested:** Apps can ask another trusted app (an identity provider) for a user's details in a `'fdc3.user'` context by **raising a standard** `CreateIdentityToken` **intent**. The identity is then shared over a secure, private connection to ensure confidentiality.

These changes aim to make app-to-app communication in financial services both **more secure** and **trustworthy**, without requiring every developer to manage the complexity themselves. See Appendix B for a technical breakdown of how this works in practice.

New Roles For Applications

By allowing user and app identity, we change the landscape of FDC3 security. We envisage that there will be two types of application, identity providers (**IDPs**, who respond to `CreateIdentityToken` requests and return `'fdc3.user'` contexts) and identity consumers (those who raise the `CreateIdentityToken` intent). Let's consider how this change plays out when a workflow happens over FDC3 enabled applications.



For a given operation:

- Users can either be *unknown*, *known* or *trusted*.
- Applications can either be *unidentified*, *identified* or *trusted*.

Let's go through each in turn to understand the roles applications now play.

User Trust Level

By default, a user can start an app on the FDC3 platform. It's always been possible for the app to manage its own user authentication, but now it has another option: it can raise the `CreateIdentityToken` request to discover a user known to an IDP. This allows the application to move the user from *unknown* to *known*.

User Level	State	Operations At This Level
Unknown		ViewNews
Known	Authenticated	View/Store User Preferences
Trusted	Authorized	StartOrder, View Trade Details

Some examples of operations requiring different levels of user trust.

To move from *known* to *trusted* requires **authorization**. This can either be handled by the app or by the IDP.

App Authorization

In this case, the app maintains a list of users and the roles they can perform, deciding on its own whether a particular task is authorized for a user.

IDP Authorization

In some JWT Claim systems, the token is sent with a **scope** field giving details of the functions the user is allowed to perform. The app can choose whether or not to trust the IDP for this granular level of control.

App Trust Level

In prior versions of FDC3 (and for applications not opting into FDC3 Security and declaring a public key), messages arriving from the desktop agent are in an *unknown* state as regards their application of origin.

However, by providing the authenticity metadata as described above, the receiving app can move them to the *known* state, resolving their identity by way of their public key information and TLS certificates. Depending on the operation, this may be enough: for audit purposes, it might be sufficient to know the identity of the user and the calling app.

App Level	State	Operations At This Level
Unknown		ViewNews
Known	Authenticated	StartCall, ViewChat
Trusted	Authorized	CreateIdentityToken, SendChatMessage

Often, when data is being returned in FDC3 (like in the Morgan Stanley / Wellington demo example with liquidity information) you not only need to know the calling app but also trust them. To move from *known* to *trusted* requires authorization. This can either be handled by the app itself (as above with user trust) or using a third party (e.g. FINOS) as described below.

App Authorization

The app maintains a list of the public key URLs / operations which it trusts. In many cases, it may augment this with additional requirements on user trust. For example, it might be necessary to assert that not only is the request from a specific *app*, but that the user has the elevated privileges required. CreateTrade, ViewLiquidity, ViewPrice are all examples where this might be true.

Third Party Authorization

For some operations, an app may want to defer authorization to a third party. In this case, the app doesn't directly know the sending app, but it does recognise and **trust one of the circles** the app belongs to (via either **certificates** trusted by the app developer or an

externally-administered **allow list** approved by the app developer). Therefore, the operation is permitted. (See Appendix C for more information).

This could be the case if a firm is delegating their Identity and Access Management (IAMS) to one of the FINOS IDP providers. This is beyond the scope of the changes to the standard but we'll cover a special case of this in the section below.

Prompt The User

A third approach is to ask the user. In this case, an app receives a message, checks the signature and identifies the app, but then **asks the user** whether the user trusts the app to do the operation.

Benefits

Let's review the issues outlined earlier and see how they are impacted by this change:

1. Shift to Web and Increased Security Needs

- Apps now have a way to encrypt data between each other.
- Apps have a trust model to use to decide when to send data to one another.

2. App Identity Limitations

- Apps have a URL (public key endpoint) indicating their identity.
- Apps are now identified on a per-broadcast basis by their (certificate-verified) URL

3. Insufficient User Authentication and Authorization

- Users are identified in FDC3 by IDPs
- User identity is portable across different FDC3 applications
- User identity becomes a first-class citizen of FDC3 via the `'fdc3.user'` context object.

4. Lack of Data Integrity Assurance

- Provenance information is included in each FDC3 message.
- Encryption is provided.

- App certification and Circles-of-Trust provide some degree of security around accidental data disclosure.
- User identity provides the ability to audit user activity in a cross-app way.

5. Pre-Existing Bilateral Trust Relationships

- Circles-of-Trust provide an optional way of avoiding re-introducing bilateral relationships back into FDC3.

6. Need for Evolution in Security Frameworks

- Even on the web, users are accountable for their actions
- An end to the “hundreds of logins” problem

Adoption Path

The adoption path we foresee is:

1. The FDC3 community works to develop FDC3 Security 2.0, containing the new features needed above.
2. IDPs create apps in the FINOS App Directory, exposing their `CreateIdentityToken` handlers.
3. FINOS certifies these early adopters.
4. MorganStanley / Wellington / StateStreet adopt the FDC3 Security libraries and join the early FINOS circle-of-trust, demonstrating secure interoperability across heterogeneous desktop environments at OSFF.
5. FDC3 2.3 contains the changes to the standard described above.
6. FDC3 Security is added to the FDC3 repo as an experimental part of the standard.
7. Other firms wishing to add user identity to their apps opt-in and join the FINOS circle-of-trust, passing app conformance. This might be useful for license management, for example.
8. Apps not requiring login (or demos) either remain unconformant or pass conformance as a security signifier but don't use `'fdc3.user'`.

Conclusion

The proposed security enhancements for FDC3 represent a significant evolution in enabling **safe, reliable, and scalable interoperability** across financial applications—especially as the ecosystem transitions from desktop containers to a web-based environment. By integrating

robust cryptographic mechanisms, such as digital signatures, encrypted channels, and JWT-based user identity, the design addresses critical challenges including application spoofing, data leakage, and unauthorized access.

The introduction of Circles of Trust, alongside clear roles for identity providers and consumers, offers a framework to bypass the limitations of bilateral trust relationships while ensuring consistent and verifiable authentication.

Although the increased complexity—**particularly in key management, certificate handling, and performance considerations**—poses challenges, the reference implementation and future refinements should help mitigate these issues.

Appendix A: Security Issues Being Addressed

Web Security Challenges due to “FDC3 On The Web”

1. Application Spoofing & Impersonation

Since web applications run in open environments, **malicious sites or browser extensions could masquerade as legitimate FDC3-enabled applications**. Without proper identity verification, an attacker could send rogue FDC3 intents, manipulate data, or hijack workflows.

2. Cross-Origin Attacks (CORS & Cross-Site Scripting - XSS)

Web applications operate across different origins (domains). This opens the door to **cross-origin request forgery (CSRF) and cross-site scripting (XSS) attacks**, where a malicious site could inject scripts into a trusted FDC3 application to execute unauthorized actions.

3. Unauthorized Data Access & Leakage

FDC3 is designed for **seamless data sharing**, but without a robust access control mechanism, sensitive financial data could be exposed to unauthorized applications. A compromised or untrusted app could **subscribe to intents, intercept context data, or exfiltrate sensitive information** across different applications.

4. Session Hijacking & Token Theft

Since web applications rely on **browser-based authentication mechanisms (cookies, OAuth tokens, JWTs, etc.)**, they are vulnerable to **session hijacking, token replay attacks, and phishing scams**. If an attacker steals an authentication token, they could impersonate a legitimate application and misuse FDC3 permissions.

5. Lack of Secure App Discovery & Trust Management

In desktop containers, applications are often pre-approved and registered, ensuring a trusted environment. On the open web, **there is no built-in mechanism to verify whether an app should be trusted** to participate in an FDC3 workflow. This could lead to rogue applications injecting themselves into communication flows, potentially compromising security.

Issues with Cross-Firm Interoperability

As evidenced in the Wellington / Morgan Stanley interoperability proof-of-concept in 2023.

1. Lack of Identity Verification

In the demo, data such as **position data** was shared between different applications using FDC3 intents. However, one of the challenges was that there was **no robust identity management** system in place to verify who was sending the data.

2. Data Integrity Risks

Since FDC3 relies on **open standards for data sharing**, without strong **authentication** and **authorization mechanisms**, there is a risk that data could be intercepted, altered, or tampered with during transmission (e.g. by the desktop agent)

3. Absence of Secure Communication Channels

FDC3, in its earlier implementations, didn't define strong enough encryption or secure communication protocols for data being exchanged between applications. Without **end-to-end encryption** and **trusted identity assertions**, organizations were left exposed to potential **man-in-the-middle** attacks and unauthorized data access.

4. Compliance & Regulatory Concerns

For firms like Wellington and Morgan Stanley, ensuring **compliance with financial regulations** (such as MiFID II, GDPR, or SEC guidelines) is crucial. The inability to ensure **data authenticity and integrity** over FDC3 created significant barriers to trust, especially when sharing sensitive financial information. Any leakage or misrepresentation of data could result in **severe compliance violations** and reputational damage.

Issues With Apps Delivering User Identity

As evidenced by Symphony's Hackathon IDP demo

1. Token Exposure and Data Leakage

JWT tokens typically contain sensitive **user information** (such as user IDs and roles), and sharing them over an open protocol increases the risk of them being **intercepted**.

2. Increased Attack Surface

By passing **authentication tokens** between applications, the **attack surface expands**. Any application that receives the token could be a potential point of compromise if it doesn't have proper access controls or if it is compromised.

3. Trustworthiness of Apps

In a demo scenario like this, it's important to consider whether all participating apps in the FDC3 ecosystem can be trusted. Without proper **app authentication** and **source verification**, there's a risk that one of the applications could be **rogue** or **compromised**, and they could misuse the JWT token to gain unauthorized access.

Appendix B: New Standard Mechanisms

1. `__signature` metadata

When FDC3 Security sends a context, it *signs* the message with the broadcasting app's identity, by adding some metadata to the broadcast payload like this:

JavaScript

```
{
  "__signature" : {
    "digest": "<the signature, encoded using the app's private key and the
contents of the context being broadcast>",
    "publicKeyUrl": "<URL of the app's public key (matching the private key)
that can be used to check the signature" ,
    algorithm: { // currently state-of-the-art, could change in the future.
      "name": "ECDSA",
      "hash": "SHA-512",
      "namedCurve": 'P-521'
    },
    "date": "timestamp, when the message was signed. Messages are expected to
have a lifespan and to avoid replay attacks."
```

```
}  
}
```

2. authenticity metadata

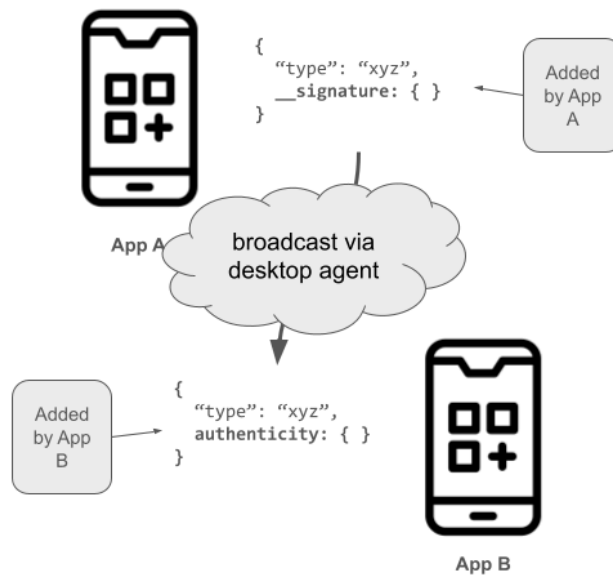
When a message is received, FDC3 Security checks the signature and adds the authenticity key to the ContextMetadata like so:

JavaScript

```
{  
  "authenticity": {  
    "verified":true,           // set to true if the broadcast  
                               was signed and had a signature that could be verified  
    "valid":true,             // set to true if the public key  
                               was able to verify the signature  
    "publicKeyUrl":"/sp1-public-key", // the URL of the JWK Set used to  
                               verify the signature, as given by the sender  
    "trusted": true           // set to true if the receiving  
                               app and the broadcasting app both belong to the same circle-of-trust.  
  }  
}
```

Any *signed context object* will have its authenticity checked and this key filled in. This information is available when an app sets up a `BroadcastHandler` and is passed along with the context object itself.

A signed context object is one which has a `__signature` key added to it, containing a digest of the context object signed by the broadcasting app's own private key.



In order to sign its requests and broadcasts, an app needs to set up a public / private key and publish its public key in a JWK Set file ([JSON Web Key Set](#)). The JWK Set is a set of public keys. This allows for old keys to be rotated out and new keys to be added over time.

By specifying the public keys at an HTTPS endpoint, the application gives a certificate-backed identity for itself which cannot be spoofed by other applications.

Finally, if the app opts in to a specific circle-of-trust, the final field "trusted" indicates whether both the sending and receiving apps belong to the same circle of trust. More details on this and the reason for having it are in the "FINOS Circle of Trust" section. It is entirely optional whether an app decides to observe this field.

3. Encryption Primitives and Request / Response Context Types

FDC3 Security adds the `'fdc3.security.symmetricKey.request'` and `'fdc3.security.symmetricKey.response'` context types, which allow members of a private channel to negotiate a symmetric key for encrypted communications on that channel.

JavaScript

```
export type SymmetricKeyResponseContext = Context & {  
  type: 'fdc3.security.symmetricKey.response',  
  id: {  
    publicKeyUrl: string  
  }  
  wrappedKey: string,  
  algorithm: any  
}
```

```

}

export type SymmetricKeyRequestContext = Context & {
  type: 'fdc3.security.symmetricKey.request'
}

```

This is intended to be used in conjunction with the authenticity data. When a data-publishing app receives a `'fdc3.security.symmetricKey.request'`, it uses the authenticity metadata to decide whether or not to broadcast a symmetric key over the channel. As the symmetric key is encrypted with a single app's public key, only the requesting app can decrypt and therefore use the symmetric key to further decrypt communications on that channel.

When an app broadcasts encrypted context data, only the type field of the context remains visible. This is necessary as the type information is used by the desktop agent and the context handlers for deciding whether to route or process the context. The rest of the context information is hidden in a field called `__encrypted`, replacing all the other fields until the context is decrypted again.

If an app receives a context object containing an `__encrypted` field, then it can choose to send a `'fdc3.security.symmetricKey.request'`, asking for the key to decrypt.

Note that all of this behaviour is handled within the FDC3 security code: apps just need to call `setChannelEncryption()` on a private channel when they want to start using encryption and then all the other behaviour is handled for them.

4. `fdc3.user` Context Type

For identity consumers to receive the details of the user, we need a new FDC3 context type, `'fdc3.user'`, which looks like this:

```

JavaScript
{
  "type": "fdc3.user",
  "id": {
    "emailAddress": "john.doe@somebank.com",
    "name": "John Doe"
  },
}

```

```
"jwt":  
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MTIzNDU2Nzg5LCJuYW11IjoiSm9zZXBoIn0.OpOSSw7e485L0P5PrzScxHb7SR6sA0MRckfFwi4rp7o", // original JWT  
}
```

Contents of 'jwt' Field

JWT or JSON Web Token is an internet standard for passing around authentication credentials, and is a base-64 encoded string which is cryptographically signed. The string is split into three sections, delimited by '.' characters. The first section is a header, telling us the format of the JWT. ie.

```
JavaScript  
// first part of the JWT token. This is standard and won't change.  
{  
  "alg": "HS256",  
  "typ": "JWT"  
}
```

The second part is the payload, like so:

```
JavaScript  
{  
  "iss": "https://symphony.com", // organisation issuing/signing the  
  identity  
  "sub": "john.doe@somebank.com", // identity of the person  
  "aud": "https://the.requesting.app/publicKeyUrl", // app asking for identity  
  "exp": 234532445325, // expiration date.  
  "jti": "32812-23423-3123" // unique ID for the key.  
  "scope": "read:profile write:operation" // things the IDP says the user can  
  do.  
}
```

By providing all these fields, we scope the JWT token between a single identity provider (IDP) and a single identity consumer (app). This is important in case tokens are leaked (more on that later).

The final part of the JWT is a signature. In much the same way as a signed context, this allows the receiving app to check the validity of the token that it's been sent. An app can prove that the token is from a particular source. In our case, we would expect the signature of the token to match the application's public key in the same way we do for the authenticity metadata.

5. Requesting User Identity

In order to request an identity, an App must raise an Intent, `CreateIdentityToken`. This is paired with a piece of context data `'fdc3.user.request'`:

```
JavaScript
{
  "type" : "fdc3.user.request",
  // space for other data
}
```

Resolving the intent follows normal FDC3 rules: if there are multiple apps that listen for `CreateIdentityToken`, then the user chooses which app to resolve with. In the case of a single app, this will be used. The single app case is the expected use-case.

Rather than broadcasting the token in public to all the applications, it is expected that the identity providing app (the IDP) will open a private channel with the requesting app and deliver the token in encrypted format over that channel. The requesting app then decrypts as described above.

Appendix C: A FINOS Circle of Trust

As described in the problem statement:

- We don't want to end up with a system of bilateral agreements again.
- We don't want a situation where untrusted apps exfiltrate sensitive information.

However, this poses a problem. Within the arrangement above we have a role for IDPs responding to `CreateIdentityToken`, but we mustn't allow just any app to be able to use this. Only *trusted* apps should be able to get identity tokens, otherwise bad-actor apps can potentially discover information about who is using FDC3.

For this reason, we would like to create a circle of trust in which:

1. **FINOS is able to admit new IDPs who will respond to `CreateIdentityToken`.**
FDC3 directory assemblers are then free to include those IDPs in their app directories.

2. **FINOS performs an app conformance process** (described below). Apps passing the conformance process will be permitted by IDPs to request `CreateIdentityToken`. FDC3 directory assemblers are then free to include these apps in their app directories, along with any others they may want to include.
3. **Apps within the circle of trust shouldn't need to know about one another** therefore, avoiding the bilateral agreements, or storing details of each other when the app is created
4. **Apps can prove their shared membership of a circle when needed.**

By this measure, we can exert some level of control over which applications have access to user information. There is prior art in this space in the form of the [Symphony App Authentication / Circle of Trust Model](#).

It's worth pointing out that this doesn't preclude other organisations adopting FDC3 and creating a similar system in their own industry. This would be welcomed. In fact, apps may want to enter into other trust relationships and manage these circles too.

Workflow For Applications Requiring Identity

1. App A joins the FINOS circle of trust. They are issued a digital signature composing their URL, joining/expiration dates and some proof from FINOS that they are part of the circle.
2. App B is an IDP, and part of the FINOS circle of trust already.
3. App A raises `CreateIdentityToken`
4. App B observes the request and notes the authenticity metadata. If it is **trusted**, it responds with an `"fdc3.user"` context object, over an encrypted channel as described above. In order that the app is trusted, App A must provide its signature to App B as part of the context data. I.e. As well as signing the context object, App B would also provide its certificate of authenticity, proving that it is a member of the same circle of trust as App A.
5. App A's FDC3 security middleware would construct the authenticity metadata for the returned context object, checking that it is signed correctly. After setting **"verified"**, **"valid"** and the **"publicKeyUrl"** it would then need to decide whether the App B is **"trusted"**. To do this, it would check the certificate sent by App B to make sure that it is signed by FINOS (in this case).

In the above example, each app checks that the other belongs to the circle of trust. That way, App B can ensure it isn't leaking user information, and App A can ensure that it's being given user information it can rely on.

Implications

- The above description is for a specific FINOS circle of trust, however, the standard should not prevent other groups from existing.

- FINOS will have to issue certificates in this manner which applications could either contain built-in or load from servers at runtime.
- Apps don't necessarily need to belong to trust groups (or the FINOS trust group) but could instead construct their own, separate from the decisions made by directory compilers or the desktop agent.
- The desktop agent remains untrusted.

The choice of this approach entirely belongs with the app, but the enabling factor is that apps must provide their identities.

Application Conformance

FINOS has experience in administering an FDC3 Desktop Agent conformance program. We would expect that a similar process of “badging” can be rolled out for applications wishing to join the initial FINOS circle-of-trust.

Conformance is not so clear cut as with a desktop agent: there is no set of functional tests you can run to ensure it works as expected. However, we would expect at a minimum:

1. **The app is using FDC3 security** and signing contexts it broadcasts.
2. **The app is running on public infrastructure** with a public URL, high-strength certificate, up-to-date HTTPS etc,
3. The app passes some basic automated security checks.
4. The app publishes its public key in the correct JWK Set format on a domain that reveals ownership via its name and certificate.
5. The app is publishing an accurate AppD record.
6. The app makes some use of `CreateIdentityToken`.
7. App developers declare what they are using identity data for in their paperwork.

If the app is an IDP, we would also expect:

1. That there is an endpoint available to FINOS where we can check the list of allowed `CreateIdentityToken` receivers, update the list with new applications as they are added to the circle
2. This endpoint is secured by password or other security means.
3. There is an audit log of changes to this data which is sent to FINOS on a regular basis.

Appendix D: Some Criticisms / Drawbacks

Complexity and Adoption

The design introduces significant complexity—particularly around key management, certificate handling, and maintaining the circle of trust. While these are essential for robust security, they could be challenging to implement consistently across the diverse financial ecosystem. As it stands, applications are required to manage:

- **Their Public Key:** To be published as part of a JSON Web Key Set (JWKS), allowing other applications to verify digital signatures.
- **Their Private Key:** Which must be securely stored and used only in controlled, server-side operations.
- **Certificates for Any Circles of Trust They Belong To:** These certificates are critical for proving membership in a trusted group (e.g., the FINOS Circle of Trust).

To mitigate these concerns, we already have a reference implementation available at <https://github.com/finos-labs/fdc3-security>. This implementation should be further developed with example apps and enhanced documentation that demonstrates proper key management practices. Notably, signing should occur on the server side—addressing the current security weakness of performing these operations client-side, which exposes private keys to untrusted environments.

Furthermore, the current implementation does not yet include complete circle-of-trust logic. This functionality needs to be added and demonstrated in a straightforward manner to show how apps can dynamically verify shared trust without resorting to bilateral agreements.

Key Management and Revocation

Apps must manage their public and private keys carefully. If a private key is compromised, the application must issue a new key pair and update its public key in the JWKS as quickly as possible, effectively retiring the old key to minimize risk. A clear strategy for key rotation and revocation is essential to maintain trust over time.

User vs. App Identity Distinctions

The expectation is that most apps will not need to differentiate heavily between which app is sending data; instead, they can rely on user privileges and the established app trust levels. This approach simplifies the trust relationships by focusing on user authorization rather than granular app identity. In directory configurations, typically only one Identity Provider (IDP) will be present—even though we anticipate multiple IDPs joining the FINOS Circle of Trust. There is also a need to refine how user roles and claims are managed at the IDP level within FDC3 to ensure consistent authorization across applications.

Performance and Latency Considerations

Cryptographic operations, including signing, verification, encryption, and decryption, can introduce latency. In the existing demos, these operations are performed on the client side, which is a security risk since it requires transporting private keys into an untrusted environment. We need to conduct performance benchmarking when these operations are moved to the server side to ensure that the additional security measures do not adversely impact the user experience. Additionally, if context objects are transmitted in large volumes, we may need mechanisms to optimize or selectively omit signing for certain types of data to reduce overhead.

Interoperability and Legacy Systems

This design is intentionally opt-in, meaning that existing applications and desktop agents can continue to interoperate without modification. However, it is important to note that some FDC3 designs, particularly those implemented by certain vendors, include interception and modification of contexts as they pass through the desktop agent. Such behavior would conflict with the integrity provided by digital signatures, as any alteration of the signed data would invalidate the signature. Therefore, the proposed design assumes that applications adhere to a model where the desktop agent is not trusted to modify context data, ensuring the validity of the digital signatures remains intact.