
*Datica Research, LLC*

Release 1.0: 20 March 2025
*White Paper*

WHITE PAPER

# Scaling Confidential Computing for Enterprise Applications: Challenges and Opportunities

John L. Manferdelli(NAE), Paul E. England (NAE) and Tho H. Nguyen*

*To whom correspondence should be addressed: helloDR@daticaresearch.com

## Abstract

Confidential Computing (CC) has gained popularity in recent years to secure data and computation, especially for sensitive applications in the cloud. Compared with other security approaches, CC demonstrates distinct advantages such as delivering high security with minimal performance impact. However, CC is not yet a broadly accessible solution and more work is needed to scale its implementation beyond bespoke applications. Future work to address accessibility and scalability include abstraction frameworks to *hide* hardware and cryptographic complexities, new tools and middleware to support large scale deployments, and platforms with built-in CC capabilities for general purpose applications. Datica Research principals, who were early contributors that led the invention of Confidential Computing, are working on scaling CC to meet today's enterprise security challenges.

**Key words:** confidential computing, trusted execution environment, cryptography, privacy-preserving, cybersecurity

## 1. Confidential Computing Fundamentals

Since the 1990s, CPU and platform vendors have added specialized security capabilities for enhanced compute and data security. This has included key stores and cryptographic co-processors, secure boot, and facilities for Trusted and Confidential Computing. The excitement for Confidential Computing is understandable due to its many advantages over related security approaches.

Confidential Computing is a mechanism that provides isolation, measurement, secret storage and attestation based on hardware primitives - these are known as the "Foundation Properties" of CC. These capabilities allow a well-written[1] program to guarantee adherence to operational (e.g., security and safety) policies as well as ensuring the confidentiality and integrity of

[1] A program with no exploitable vulnerabilities that can cause unintended behavior such as unauthorized disclosure of information to an unauthorized party, the corruption or unauthorized modification of a program or information affecting processing, or corruption of the program by introducing malware or other adversarial artifacts that can adversely affect the program. Implicitly, "unauthorized" actions or access means actions not expressly authorized under a policy set forth as the basis of the programs processing "contract" between or among users of the program and those who rely on it.

the program and the data it uses. This protection is reliable in the face of attacks by other programs running on the computer, communications over a network and even the malicious actions of the administrator or owner of the computer on which the program runs or the networks it uses.

Confidential computing derives security from two foundations: **Cryptography**, and, **immutable hardware behavior**.

### 1.1 On Cryptography

The **cryptographic primitives** employed by Confidential Computing are the common ones widely used[2].

*Cryptographic hashes*: Confidential computing elevates computer programs to first-class security principals for access control and authentication. To enable this security, programs must be reliably identified. The most common authentication mechanism is the cryptographic hash of a program and its configuration. Cryptographic hashes are one way functions which have the property that given an arbitrary sequence of bits, $B$. It is easy for anyone to compute the hash, denoted as $h(B)$ but it is infeasible, even given access to vast computing resources, to find a different string, $B'$, such that $h(B) = h(B')$. Hashes are useful because they provide a short "fingerprint" for large data files and programs.

*Symmetric ciphers*: These are public algorithms denoted by a pair of functions, $E_K(M)$ and $D_K(C)$ that allow two parties (often abstractly called Alice and Bob) who share knowledge of a secret key, $K$, (whose size is about 256 bits, usually) to communicate over public channels which can be observed by adversaries so that only they can "read" the protected information. For a message, $M$, $C = E_K(M)$, is called the ciphertext and $D_K(E_K(M))=M$. Symmetric ciphers are very fast and can process huge messages in microseconds. Again, without knowledge of the key $K$, even given access to vast computing resources, no unauthorized party can discover the encrypted text or any part of it.

*Asymmetric ciphers*: Asymmetric (or public key) ciphers consist of a pair of functions $(E_P(M), D_S(C))$. Again, $D_S(E_P(M)) = E_P(D_S(M))=M$. $P$ is called a public key and $S$ is called the secret or private key. However, in asymmetric ciphers, while only the owner knows the secret key $S$, anyone can know the public key $P$. This means anyone can encrypt messages that only the holder of the private key, $S$, can decrypt. Even with knowledge of $P$ and the algorithm, and even given access to vast computing resources, $S$ is not computable. And of course, without knowledge of $S$, even given access to vast computing resources, without $S$, no one can decrypt $E_P(M)$.

Asymmetric ciphers are most commonly used in two ways: (a) to allow Alice to share a secret key with Bob, and (b) to allow Bob to digitally "sign" a message that Alice can verify. In other words, cryptographic hashes and asymmetric ciphers can be employed to produce *unforgeable* signatures that can be verified by anyone. This is done as follows. Suppose one wished to sign an Agreement, A, consisting of a sequence of bytes (like a word document). The signed computes $D_S(h(A))$. Given $A$, and $P$, anyone can verify whether $E_P(D_S(h(A))) = h(A)$ but only the secret key holder can compute $D_S(h(A))$. Asymmetric ciphers can be used to "authenticate" or verify the identity of a secret key holder (provided you know for certain that the public key, $P$, is associated with the entity you wish to authenticate) by issuing a random challenge, $M$. The secret key holder then computes $D_S(M)$ and the person issuing the challenge can verify that $E_P(D_S(M)) = M$. Asymmetric ciphers can also be used to encapsulate symmetric keys, anyone can generate a key $K$ and send it to the secret keyholder by computing and transmitting $E_P(K)$. However, usually, the most useful form of key transmission is "authenticated key exchange" where Alice and Bob exchange symmetric keys, using asymmetric ciphers, confidentially, with the additional assurance that Alice can be sure Bob is the only other party who got the key and Bob is sure that Alice is the only party that got the secret key. We won't describe the construction of authenticated key exchange using asymmetric ciphers but they are well known. Cryptographic hashes in conjunction with symmetric ciphers can not only ensure the confidentiality of a message but also its integrity so that even an adversary that can modify or add bits during the transmission of a message can deceive the rightful receiver of the message into accepting any message or part of a message other than from the rightful transmitter.

The primitives above and the artful use of them provided, with the hardware described below, everything needed for confidential computing.

### 1.2 Immutable Hardware Behavior

Confidential computing hardware platforms usually uses the cryptographic primitives to provide the Foundation Properties as follows.

1. When the program is read into main memory, the hardware (atomically) computes its cryptographic hash (we'll call this the program *measurement*, $m_P$)[3]. It then *isolates* the program using the memory management unit and protect against various hardware attacks, often by encrypting the program memory and registers with a unique integrity and confidentially-protecting key. No other program or even the OS has access to these keys. While the program runs, the measurement, $m_P$, is unforgeably associated with the program by the hardware.[4]

2. The hardware can provide *secret storage* by using a confidentiality and integrity protecting key, $K$, that only the hardware knows. A program can ask the hardware to save a secret, $S$, the hardware encrypts and integrity protects the bit string $(S, m_P)$ perhaps with a nonce and returns the encrypted bits, $C$, to the program. If the program needs to recover $S$ (say, after it restarts), it requests the secret from the hardware by supplying $C$. The hardware decrypts and integrity verifies the decrypted string $(S, m_P)$. If $m_P$ is the measurement of the running program, the hardware returns $S$. This mechanism is usually called *sealing* and is commonly used for offline data storage.

3. Finally, there is *attestation*. An attestation is a message, $M$, unforgeably linked to the program measurement, $m_P$,

[2] These primitives are described in great detail in texts, standards, research papers and internet sources, such as Wikipedia.

[3] The hardware may also add any program affecting data and properties when computing the hash to guarantee these as well.

[4] Many platforms - particularly smaller systems - perform program measurements, but do not provide isolation for mutually distrustful modules. Such systems can be treated identically to larger systems, but typically can only run a single CC program.

produced by the hardware for which a remote party can securely verify that $M$ comes from the program $m_P$ while isolated and protected. The easiest way to do this is to equip the hardware with a public, private key pair ($P$, $S$) accompanied by unforgeable evidence that $P$ is known only to this particular hardware. The attestation is then the statement, signed by $S$, "The program, $m_P$, isolated by my protection, said $M$." The common use of attestation is to securely introduce a program public key, $P_{program}$, whose secret key $S_{program}$, is known only to the program and accessible only when protected. The meaning of that attestation is "The program with measurement, $m_P$, says that $P_{program}$ is its authentication key and only the program knows the key $S_{program}$. The upshot is that once this attestation is verified, "relying parties" (the parties to whom the attestation is sent), can be sure that any entity proving possession of $S_{program}$, must be the program while protected. (Proof of possession is another simple set of protocols using asymmetric keys). Two programs *program1* and *program2*, once proof of possession is established of their keys $S_{program1}$ and $S_{program2}$, $P_{program1}$ and $P_{program2}$ can be used to establish authenticated, encrypted, integrity protected channels between them (much like "mutually authenticated TLS channels").

As a result of all this, once you know that *program1* and *program2* are well written and you know exactly how they process information, you can compute their measurements, $m_{program1}$ and $m_{program2}$ and use the primitives above to guarantee their isolation, the confidentiality and integrity of the data they use and the integrity of their program flow. This protection holds even if the program is running on a machine you don't control (say in a cloud data center), even if there is malware on those machines and even if the administrator (the "root" on that machine) is malicious. This is a rather strong, principled security guarantee that provides rather comprehensive and remotely verifiable assurance. It also simplifies and secures key management. Finally, programs so protected can be produced by any program written in any language by people who understand the "API" of the program. Further, except for the nearly negligible time to set up and save program keys, these programs can run (depending on the platform) at "full CPU speed."

## 2. A Comparison Across Different Secure Computation Approaches

There are several related security techniques with different protections and capabilities than the ones Confidential Computing offers:

*Differential Privacy* (DP) is a technique that limits disclosure of individual data items by responding only to large aggregate queries and "adding noise." It is not intended to protect the storage or query processing of the data only the output of aggregate queries. Good application: Census data.

*Fully Homomorphic Encryption* (FHE): Once data is encrypted (which must happen at a trusted location), the computations can be distributed to untrusted machines and processing occurs on encrypted data. Increases computation time by very large ratios that grow with size of computed data.

*Multi-party Computation* (MPC): Once data is encrypted (which must happen at a trusted location), computations can be distributed to untrusted machined and processing occurs on encrypted data. Computations are limited and require custom programming by experts but computation time although much higher is much lower than with Fully Homomorphic Encryption. Generally protects small bespoke applications where limited collusion provides protection.

The common theme among these approaches is that they rely strictly on algorithms (and often cryptography) to provision the security properties[5]. This results in rather severe restrictions on their applicability and scalability. MPC and FHE are generally only effective as custom solutions for small applications. And DP is only effective for large databases to prevent cross referencing (and DP algorithms must coordinate with the analytics such that the noise can be effectively filtered out).

Unlike algorithmic approaches, *Confidential Computing* leverages both crytographic and hardware primitives to provision secure enclaves where it's unnecessary to inject noise into the data (compared to DP), and no need to encrypt the data and algorithms beforehand (compared to

[5] Recent hardware progress aims accelerate the implementation of these approaches, e.g., for FHE, but they hardware is not fundamental to their security model.

**Table 1.** Comparison between secure computation approaches.

| Approach | MPC | FHME | DP | CC |
|---|---|---|---|---|
| Security model | Protect encrypted data during computation on redundant platforms that cooperate | Protect encrypted data during computation | Prevent query from returning individual data | Protect data and computation from inception to storage and use |
| Threat model | Protects data after encryption | Protects data after encryption | Protects individual data items in large aggregate queries. Does not protect from insiders and malware | Protects granular data, no restriction on data size or program size |
| Performance impact | High | Very high | Moderate | Negligible |
| Scalability | Small custom applications | Small custom applications | Aggregate (large) DB queries only | Any application |
| Iplementation Complexity | Very high | Very high | Moderate | Very low |

MPC and FHE)[6]. This means existing applications need not be significantly rewritten to leverage Confidential Computing[7], and it is also very possible to scale confidential computing to support large-scale enterprise systems or a general purpose application platform.

A straightforward comparison between different secure computation approaches is given in **Table 1**.

Here are some applications in reach for Confidential Computing:

- Cloud security enablement
- Hardware secure module
- Secure Key Store and token generation
- Standard platform components (storage, time, IAM)
- Secure shared database access
- Secure privacy preserving service enablement
- Secure Motion planning as a service
- Secure collaborative machine learning
- Secure Auctions
- Secure infrastructure management
- Secure Kubernetes container management (via secure Spiffie/Spire)
- gRpc and Zero Trust
- Secure Document sharing
- Secure "cradle to grave" data provenance
- Edge sensor collection
- Enabling common platforms for sensitive edge services
- Caching services and the "extended internet"

There is currently no technology to convert a large, vulnerable program, Such as MS Word, into a secure program that can be unconditionally protected by CC, or any other secure computation approaches. This overcome this gap, several challenges must be addressed to scale Confidential Computing.

**A note on "Better Together":** Of course, *confidential computing* can be combined with these other techniques. For example, CC can protect data and storage for MPC and FHME when initially encrypted. Similarly, CC can protect the data and processing in DP during the query processing and before query response. MPC and FHME can be used with CC for extremely sensitive computations thus adding resilience to single points of hardware compromise (Byzantine agreement and key splitting can also be used to achieve this result).

## 2. Challenges to Scaling Confidential Computing

In principle, Confidential Computing can be used to protect any program. The hidden limitation is that the basis of protection is the correctness of the program. Additionally, while confidential computing protects programs from reading or modifying protected memory, it does not stop a malformed input from causing the program to misbehave or reveal secrets. I.e., while confidential computing stops attacks "from the bottom" - e.g. from the OS or other programs running on the platform, it does not help with the much more common security challenge of network or other attacks. And finally, as programs grow in size it is time consuming to build enough components safely enough to meet CC's full value.

To date, Confidential computing has been used successfully the build small programs such as *soft Hardware Secure Modules* and *Key repositories*. These are small enough that developing the underlying program safely enough is easy. Access control programs can also be built. In fact, enough infrastructure can be built to support analysis of shared data while protecting the right of each individual data owner.

Analysis of secure components can be time-consuming but a general problem is *ensuring the secure composition of CC protected components as well as CC protected components with non-CC protected components*.

In addition, analyzing components and their composition, generally requires complete knowledge of the program (i.e.-all source code). Proprietary programs must be analyzed under a protection regime that meets the needs of the IP owner.

For Confidential Computing to support enterprise applications, these fundamental gaps must be addressed.

## 3. Future Research and Development Directions

Future R&D for Confidential Computing is informed by where it is needed[8]. First, bespoke applications collecting and handling sensitive data need to integrate CC security capabilities into its data lifecycle. In order to do so, the difficulties of cryptography and hardware behaviors (and differences across hardware platforms) need to be abstracted away. Second, large scale and cloud systems require a host of middleware and tools to scale CC security capabilities to meet their needs. And finally, "general purpose" applications wishing to benefit from CC-enabled security but does not want to have to integrate CC into its stack can deploy on a CC-enabled platform.

One barrier to using CC, has been the great variety of platforms and the mountain of "standard code" needed by all CC applications as well as a mechanism to manage CC applications. This has been solved to a large extent by The Certifier Framework for Confidential Computing which provides all this as well as a standard abstract API so programs can be written and run on any platform. There are not enough Confidential Computing hardware platforms available and there is a dearth of client platforms in particular. Datica is working to remedy this.

As pointed out above, in addition to the Certifier Framework more secure middleware must be developed to cut program development time and secure composition tools must be developed along with safe access control laguages to encourage interoperability. One promise of CC is to provide "birth to death" protection of data. Development of IoT CC capability is required for this. In addition, protection of IoT and Cyber Physical components protected by CC opens a huge vista for secure data collaboration (a swarm of cars) as well as protecting factory floor and outsourced factory floor support service.

[6] It is very much possible to encrypt data and algorithms before they're passed into the enclave to enhance security. The key point here is that the algorithms need not compute on encrypted data inside the enclave.

[7] This burden is further reduced by the Certifier Framework, discussed in Section 3.1.

[8] Multiple directions for CC R&D is possible because of its flexibility and scalability potential as discussed earlier.

Finally, Datica Research (and we hope others) are building a set of "off the shelf" applications like secure messaging but there needs to be many more of these.

## 4. Conclusions

By building on foundational properties derived from cryptography and hardware-enabled, immutable security primitives Confidential Computing is able to secure workflow, deliver secure platforms, and provide tools and services to secure large scale and cloud applications. These flexibility scalability capabilities position CC to be the most promising technology for protecting computation and data; and, in combination with other secure computation approaches CC can truly meet today's enterprise security needs. Datica Research, led by researchers who pioneered the early work inventing Confidential Computing, is vigorously pursuing research efforts to build and prove CC properties at scale.

## Authors

**John L. Manferdelli, PhD (NAE)** is a principal at Datica Research. He is also Chairman of the National Academy of Sciences' Forum on Cyber Resilience, Chair of the National Academies' consensus study on cyber hard problems, and member of the Defense Science Board (DSB).

Before joining Datica, John worked at VMWare, where he was the principal designer (alongside Ye Li) of the Certifier Framework for Confidential Computing. His prior roles include Director for Production Security Development at Google and Senior Principal Engineer at Intel Corporation, where he co-led the Intel Science and Technology Center for Secure Computing at UC Berkeley (with David Wagner). He also held leadership roles at Microsoft, where he worked on Trustworthy Computing (with Paul England), and headed up Microsoft Security. Before that, he co-founded Natural Language Incorporated (with Jerrold Ginsparg), which was later acquired by Microsoft.

John's journey through computing included Bell Labs, LLNL, and TRW. Along the way, he also found time to teach mathematics and computer science at Stevens Institute of Technology, University of Washington, and UC Berkeley. He was also a member of the Information Science and Technology advisory group at DARPA.

John was elected to the National Academy of Engineering (NAE) for his pioneering work advancing Confidential Computing.

John's professional interests include cryptography and cryptographic mathematics, combinatorial mathematics, operating systems, and computer security. He is author of many papers on computer security, high performance computing, cryptography, quantum computing, computer security and signal processing and has been awarded many patents. He is also a licensed Radio Amateur (AI6IT).

John has a bachelor's degree in physics from Cooper Union for the Advancement of Science and Art, and a Ph.D. in mathematics from the University of California, Berkeley..

**Paul E. England, PhD (NAE)** is a Principal at Datica Research. Paul led or contributed to most of the computer industry's hardware-based security innovations of the last 25 years. Most notable is the field of Trusted Computing: a combination of novel cryptographic operations together with hardware/software environments for secure computation. Trusted Computing primitives are now a feature of most modern computer systems. These features are the security foundation for Microsoft (and other) clouds and are increasingly used for edge devices. In 2018 Paul was elected to the National Academy of Engineering (NAE) for this work.

Paul has worked closely with industry and standards groups to deliver the necessary hardware security building blocks. Paul led the development of the TPM specification and reference implementation, as well as co-designing many of the silicon security features we use today (TrustZone, various kinds of secure and authenticated boot, SGX-precursors, etc.) At Microsoft, he worked closely with business groups to develop features based on the hardware innovations, including a 4-year stint in Windows where Paul was the lead architect for Microsoft's disk encryption technology.

More recently, Paul led a government-industry initiative to improve the resilience and recoverability of computer systems in the face of cyber-attack as well as a minimal root-of-trust called DICE, which is becoming standard in all programmable devices: everything CPUs, microcontrollers and storage controllers, to full systems like GPUs, smart-NICs, and networking equipment.

Paul serves on several National Academy committees charged with advising on United States cyber-security policies and has advised other European countries. Paul has a Ph.D. in condensed matter physics from Imperial College of Science and Technology.

**Tho H. Nguyen, PhD** is a Principal at Datica Research. Tho started his career building data centers. With only a few wins under his belt, Tho left industry and joined the National Science Foundation. There, he helped shape the Cyber Physical Systems program in its early days. Tho returned to academia and joined the University of Virginia as a Senior Scientist in the Computer Science Department. There, he managed research in processing-in-memory and worked with the VP for IT to build out UVA Research Computing.

Since 2021 Tho has been a Senior Program Officer at the National Academies of Sciences, Engineering, and Medicine. There, he directs the Forum on Cyber Resilience, and serves as Study Director for numerous consensus studies. Topics he works on include post-exascale computing, cyber resiliency hard problems, software assurance and nimbleness, machine learning safety, Digital Twins, and semiconductor security - all of which means he's highly effective at getting people to agree on things.

Tho has a PhD (and an MS, a BS, and a BA) from the University of Washington in Electrical Engineering - except for the BA, which was for Applied and Computational Math (he can't really recall how that happened).