Towards Decentralized and Provably Secure Cross-Domain Solutions

Joud Khoury(✉), Zachary Ratliff, and Michael Atighetchi

Raytheon BBN, Cambridge, MA 02138, USA
{joud.khoury,zachary.ratliff,michael.atighetchi}@raytheon.com

Abstract. Cross-Domain Solutions (CDS) are widely deployed today for secure and timely sharing of information across security domains. Content filters are a key function of the CDS used to mitigate data threats. CDS's today are centralized and trusted and their deployments are being increasingly consolidated at the enterprise. This centralization and reliance on always-on connectivity to the enterprise introduces risk to timely and secure information sharing at the tactical edge. In this work, we take a step towards decentralizing the CDS functionality by distributing its security relevant components across untrusted tactical edge devices while still providing guarantees on the integrity of the end-to-end filtering pipeline. We instantiate a proof-of-concept decentralized CDS for bitmap image filtering and we demonstrate two alternative designs with similar trust assumptions but different performance tradeoffs. Both designs are based on *verifiable computation*. Our most performant system is able to filter a 250×250 pixel image in 15 s, $20\times$ faster than a strong baseline, and is able to scale to much larger images ($13\times$ larger scale than baseline within the available memory budget). We discuss ongoing and future work enhancing the expressiveness, performance, and security of the design.

1 Introduction

Cross-Domain Solutions (CDS) are widely deployed by the Department of Defense (DoD) for secure and timely sharing of information across security domains to support joint, interagency, and multinational mission operations. A transfer CDS filters and passes information flows between different security domains to protect against a wide range of data threats. Filters are a critical component of a CDS, performing a variety of functions such as data verification, inspection, sanitization, cleansing, and transformation to mitigate threats. Examples of such filtering include removing potential malware from the data via

Approved for public release (PA); distribution unlimited. PA Case No: AFRL-2021-1484
This material is based upon work supported by the United States Air Force under Contract No. FA8750-20-C-0200. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force.

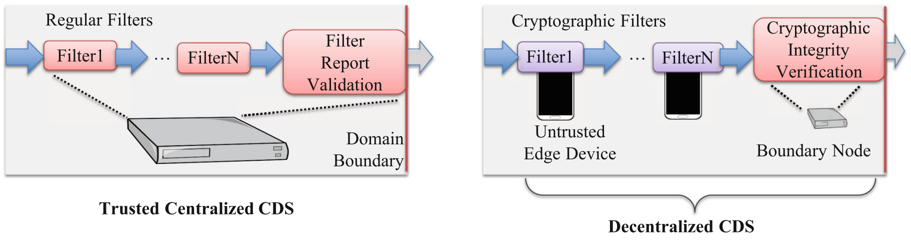


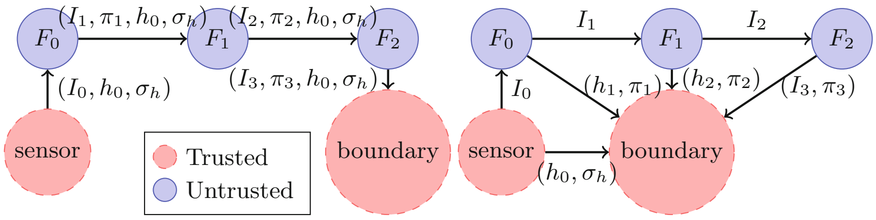
Fig. 1. A decentralized CDS distributes security relevant components over low assurance devices while providing end-to-end provable guarantees of the integrity of the filtering pipeline. This enables secure cross-domain information sharing at the tactical edge where access to a centralized enterprise or tactical CDS may not be available.

normalization, removing particular parts of the data via redaction, or transforming the data to new forms (e.g., reducing precision of coordinates or imagery or converting between protocols). The CDS ensures that the filtered data complies with the security policy. CDSs are widely deployed across the DoD, and are starting to be deployed for securing commercial operational networks [2,19].

Given that software-based CDSs are too complex to be able to prove their security mathematically, significant engineering thought has gone into successfully building CDSs that are secure by design, as defined for example by the National Security Agency's (NSA) Raise The Bar (RTB) strategy [17]. The pillars of the RTB strategy are the least privilege and least knowledge, and the Redundant, Always Invoked, Independent Implementations, and Non-Bypassable (RAIN) principles. Briefly, these principles ensure that failure of a single security-relevant component (e.g., a filter or a domain separation component) results in contained damage and does not compromise the whole CDS.

As a result, CDS platforms used by the DoD today are trusted. Their centralized and protected implementations (software and configurations) contribute to the high level of trust placed in them. Additionally, the community has started to consolidate CDS deployments on the enterprise, to gain a better picture of all existing network cross-connects. We argue that such centralization of the CDS and reliance on always-on connectivity to the enterprise introduce risk to mission success especially in Disconnected, Intermittent, Limited bandwidth (DIL) tactical environments where connectivity is intermittent.

In this work, we describe how to decentralize the CDS by distributing its security relevant components across existing low-assurance tactical edge devices while still providing the desired high-assurance guarantees required of CDSs (Fig. 1). A decentralized CDS capability complements today's enterprise and tactical CDSs enabling secure cross-domain information sharing when access to the enterprise or single-device tactical CDSs is not available. Decentralization additionally protects against the centralized CDS becoming a single point of failure or compromise. The resulting Decentralized CDS (DCDS) can initially support a limited and well-defined subset of information flows that are necessary for effective operations in disconnected tactical environments.



(a) Proof-Carrying Data (PCD) (b) Trusted Proof Aggregation (TPA)

Fig. 2. Two instantiations of a decentralized CDS filtering pipeline with three filters F_0, F_1, F_2. Sensor produces and signs images that get filtered on the way to a domain boundary (the verifier). Each filter acts as a prover generating a proof of the integrity of its local computation. The domain boundary accepts and passes the filtered images if and only if the cryptographic proof(s) π_i associated with the filtered content is/are valid; (a) proofs are recursively composed using proof carrying data and the domain boundary only needs the last proof π_3, or (b) all intermediate proofs are aggregated at the domain boundary.

A key technical challenge with decentralization is ensuring end-to-end correctness and security of the CDS filtering pipeline despite individual components of the pipeline being untrusted. Today's enterprise and tactical CDSs employ trusted operating systems and/or trusted hardware, ensuring system integrity using mechanisms such as Trusted Platform Module based trusted boot, Mandatory Access Control, Discretionary Access Control, and OS-level integrity monitoring. The strategic combination of these security mechanisms provides the basis for building the trusted filtering pipelines adhering to the RAIN principles. When decentralizing the CDS such that its components run on different edge platforms, we can no longer assume the platforms are trusted yet we must still ensure the integrity of the end-to-end filtering pipeline.

With the help of advanced cryptography and specifically *verifiable computation*, we instantiate a proof-of-concept filtering pipeline where different filters reside on different *untrusted* edge platforms, and the end-to-end integrity of the pipeline is provably guaranteed. Specifically, in our DCDS setting, a *sensor* produces and signs content (imagery in this case). The content is passed through a pipeline of compute nodes each of which performs permissible filtering on the content before forwarding it to the next node in the pipeline. The filtered content arrives at a trusted *domain boundary* (the verifier), which verifies the authenticity and integrity of the final result, i.e., it verifies that the final transformed content is a compliant transformation of the original content and is authentic, before passing the content across the domain boundary.

As shown in Fig. 2, we describe two alternative decentralized CDS architectures with similar trust assumptions but different performance tradeoffs. A key theme in our design is that every (untrusted) filtering node must furnish a cryptographic *proof* of correctness of its local filtering computation. The first architecture (Fig. 2(a)) is implemented as an application of Proof-Carrying Data

(PCD) [9,10], whereby the intermediate proofs are recursively composed in order to attest to the integrity of the end-to-end filtering pipeline. PCD is a cryptographic scheme that allows recursive composition of cryptographic proofs of computational integrity. Each party involved in the computation, receives one or more inputs I_i where each input is associated with a short cryptographic proof π_i. The party performs a local computation on the received inputs and on its local inputs and produces an output I_{i+1} along with a short cryptographic proof π_{i+1}. Given a cryptographic proof π_i, any party can verify the integrity of the full computation history up to step $i-1$. Verification is public and is very fast, on the order of milliseconds, and proofs are very short on the order of hundreds of bytes. In this PCD setting, the trusted domain boundary receives a single proof that attests to the authenticity and integrity of the entire end-to-end distributed filtering computation.

The second architecture (Fig. 2(b)) is implemented using a Trusted Proof Aggregator (TPA). Each filtering node at step i performs a local computation on the input I_i and forwards the output I_{i+1} to the next node in the pipeline for additional filtering. Additionally, node i produces a tuple (h_{i+1}, π_{i+1}) corresponding to a hash and a cryptographic proof, that it forwards to the domain boundary. The domain boundary node aggregates and checks each of the individual proofs to verify the integrity of the end-to-end filtering computation. Our TPA architecture exploits the fact that the boundary node must be trusted regardless of whether we use PCD or other designs, since the boundary node ultimately makes the decision on whether to pass content across the boundary or not. This insight allows us to aggregate the proof verification at the trusted boundary node using simple proof chaining. Proof chaining is simpler and admits a set of optimization that enhance its performance relative to the recursive proof composition design. Proof verification is a lightweight operation that can be efficiently lowered to hardware on the boundary node (such as an FPGA) and placed along with other hardware-based functions such as the diode responsible for one way transfers.

We use filtering of bitmap images as the example data type in this paper since most bitmap image formats such as JPEG, PNG, GIF, and TIFF can be converted into raw bitmaps without losing information, and in fact this is a common normalization step used by CDSs today for filtering images [17]. We implement bitmap filtering adhering to NSA's Inspection and Sanitization guide for Bitmaps [11], filtering both header and pixel data. Our most performant implementation takes around 15 s to filter a 250 × 250 pixel image for a single step of the pipeline. This is 20× faster than the baseline PCD system. Our optimizations reduce the total number of constraints by 30× allowing scaling to 13× larger images than baseline within the same memory budget. Finally, we significantly reduce the end-to-end latency of the pipeline by having the filtering nodes prove in parallel. This means our speedups relative to baseline increase with the depth of the pipeline. Our predicates can be used to filter text documents, not just images, by treating characters as pixels.

Our work makes the following novel contributions:

– Design, implementation, and performance evaluation of two alternative architectures for decentralizing the CDS that achieve provable end-to-end authenticity and integrity guarantees with different performance tradeoffs; to our knowledge, this is the first application of verifiable computation to decentralizing cross domain solutions.
– Implementation and optimization of compliance predicates for bitmap filtering. Our optimizations collectively deliver 20$\times$ speedups over prior work for image authentication from PCD [18] at the 250 $\times$ 250 image size, and 13$\times$ image scaling. In addition, the end-to-end latency of our filtering pipeline is very weakly dependent on the pipeline's depth.

The rest of the paper is organized as follows. Relevant background on cryptographic protocols is presented in Sect. 2. We present the two DCDS designs in Sect. 3, followed by the implementation details in Sect. 4, and the optimizations in Sect. 5. We evaluate the performance of the design in Sect. 6. Finally, related and future work is presented in Sect. 7 before concluding.

2 Background

We review the definitions of arithmetic circuits, preprocessing zk-SNARKs, and proof carrying data, and we refer the reader to [6] for details.

2.1 Arithmetic Circuit Satisfiability in Field $\mathbb{F}$

An $\mathbb{F}$-arithmetic circuit $C : \mathbb{F}^n \times \mathbb{F}^h \rightarrow \mathbb{F}^l$ is defined by the relation $\mathcal{R}_C = \{(x, a) : C(x, a) = 0\}$. Here a is called the witness (auxilliary input) and x is the public input and the output is 0. The language of the circuit is defined by $\mathcal{L}_C = \{x : \exists a, C(x, a) = 0\}$. Here $x \in \mathbb{F}^n$ (i.e., x is represented as n field elements), $a \in \mathbb{F}^h$, and the output in $\mathbb{F}^l$. A hashing circuit for example takes the (private) input/witness a and its hash x, and asserts that $H(a) = x$.

2.2 Preprocessing zk-SNARK

A preprocessing *zero-knowledge succinct non-interactive argument of knowledge* (pp-zk-SNARK or simply zk-SNARK) for $\mathbb{F}$-arithmetic circuit satisfiability comprises three algorithms (G, P, V), corresponding to the *Generator*, the *Prover*, and the *Verifier*.

$G(\lambda, C) \rightarrow (\mathsf{pk}, \mathsf{vk})$ Given a security parameter λ and the $\mathbb{F}$-arithmetic circuit C, sample a keypair comprising a public proving key pk and a public verification key vk.

$P(\mathsf{pk}, x, a) \rightarrow (\pi)$ Given the public prover key pk and any $(x, a) \in \mathcal{R}_C$, generate a succinct proof π attesting that $x \in \mathcal{L}_C$

$V(\mathsf{vk}, x, \pi) \rightarrow b \in \{0, 1\}$ checks that π is a valid proof for $x \in \mathcal{L}_C$.

The zk-SNARK is the basic cryptographic building block we use to instantiate both CDS architectures.

2.3 Proof Carrying Data (PCD)

Proof carrying data allows distributed computation among mutually-untrusted parties [6,9]. Each party receives s input messages each of size n from other parties $\boldsymbol{z_{in}} \in \mathbb{F}^{s.n}$, adds its local input $z_{\mathsf{loc}} \in \mathbb{F}^{n_l}$ of size n_l to it, and produces an output $z \in \mathbb{F}^n$ along with a succinct proof which is sent to downstream parties in the computation graph. Here s is referred to as the *arity*.

A *compliance predicate* $\prod$ defines the valid local computation performed at each party. Given a message z and a proof, the goal of PCD is to ensure $\prod$-compliance i.e., that every local party's computation along the sequence of computations that produced z satisfies $\prod$. The predicate $\prod$ is represented as an $\mathbb{F}$-arithmetic circuit with inputs $(z, \boldsymbol{z_{in}}, z_{\mathsf{loc}}, b_{\mathsf{base}})$ where $b_{\mathsf{base}} \in \mathbb{F}$ denotes whether the local party is the base party i.e., has no predecessors in the computation graph.

A PCD system comprises algorithms (*Generator*, *Prover*, *Verifier*), corresponding to the generator, prover, and verifier.

Generator$(\lambda, \prod) \rightarrow (\mathsf{pk}, \mathsf{vk})$ Given a security parameter λ and the compliance predicate $\prod$ expressed as a $\mathbb{F}$-arithmetic circuit, sample a keypair comprising a public proving key pk and a public verification key vk.

Generator$(\mathsf{pk}, \boldsymbol{z_{in}}, \boldsymbol{\pi_{in}}, z_{\mathsf{loc}}, z) \rightarrow (z, \pi_{out})$ Given the public prover key pk, a set of input messages $\boldsymbol{z_{in}}$ with corresponding compliance proofs $\boldsymbol{\pi_{in}}$, local input z_{loc}, and output z, generate a succinct proof π_{out} attesting that z is $\prod$-compliant.

Verifier$(\mathsf{vk}, z, \pi) \rightarrow b \in \{0, 1\}$ checks that z is $\prod$-compliant.

Appendix A reviews an instantiation of PCD from zk-SNARK and elaborates on the performance of such system in terms of circuit size and prover algorithm space and time complexity.

3 Decentralized CDS Designs

We present two designs for a DCDS filtering pipeline. The first is a direct application of PCD whereby integrity proofs are recursively composed as content gets transformed along the filtering pipeline, while the second design relies on a *trusted aggregator* to perform the proof composition. In our simplified transfer CDS scenario, the domain boundary node must be trusted as it ultimately has to make the decision on whether to pass filtered content across the domain boundary. This is true regardless of the design, whether centralized or decentralized. We leverage this inherent trust assumption in order to design the trusted aggregator, which significantly enhances the efficiency of the filtering pipeline while providing similar assurances.

3.1 DCDS from Recursive Proof Composition

Our PCD design is inspired by [18], where the authors implement a compliance predicate for image authentication under a set of permissible transformations. Figure 2(a) presents a high level overview of the approach. Consider an existentially unforgeable signature scheme $\mathcal{S} = (G_S, S_S, V_S)$ with private signing key v_s and public verification key p_s (e.g., ECDSA), and let H be a collision-resistant hash function. The sensor produces an image I_0, hashes it to $h_0 = H(I_0)$, signs the hash using its private signing key v_s to produce σ_h, and sends the tuple (I_0, h_0, σ_h) to a successor filtering node F_0. Filter F_0 performs a *permissible* filtering computation on the input image I_0 (e.g., redaction, cropping, rotation, scaling, and so on) as defined by the compliance predicate $\prod$, generates the filtered output image I_1 along with cryptographic proof π_1, and forwards the tuple $(I_1, \pi_1, h_0, \sigma_h)$ to the next filter in the pipeline, and so on. Finally, the domain boundary node checks the authenticity of the image and the integrity of the end-to-end filtering pipeline simply by checking the last proof π_3. Verifying π_3 ensures that the filtered image I_3 has a *permissible provenance* i.e., I_3 is the output of a set of $\prod$-compliant computations on an original input image whose hash is h_0, and that σ_h is a valid signature of h_0 under v_s.

The compliance predicate $\prod(z_{in}, z_{\mathsf{loc}}, z_{out}, b_{\mathsf{base}})$ for image authentication, i.e., the local computation that each node must perform, is shown in Algorithm 1. The base filtering node (which has no inputs, and has the original signed image from the sensor, and has $b_{\mathsf{base}} = 1$) verifies in the PCD that $h_{in} = H(I_{in})$ is a valid hash of the original image $I_{in} = I_0$. The base node, and every successor filtering node along the way, verify that $h_{in} == h_{out}$, i.e., the hash is passed through the computation unchanged, and verify that the output image $I_{out} == F(I_{in}, \gamma)$ is a valid filtering transformation of the input image I_{in} according to $z_{\mathsf{loc}} = (F, \gamma)$ where F is the filtering transformation identifier and γ is metadata for the transformation such as size for cropping or scaling factor.

The domain boundary verifies that,

- the PCD proof is valid, i.e., *Verifier*$(\mathsf{vk}, z_{in}, \pi_{in}) == 0$ which ensures that I_{in} is a permissible provenance and it is authentic, and
- the signature σ_h is a valid signature of h_{in} under p_s where p_s is the public verification key, and this step as in [18] is performed *outside-the-PCD* for efficiency reasons as this avoids having to implement signature verification in the PCD. More formally, it checks $V_S(p_s, h_{in}, \sigma_h) == 0$

With this scheme, only the base node runs the hashing functionality in the PCD. An efficient hashing circuit from [18] from subset-sum exists already (it is required for the PCD system itself). A hash fits in one element, and the ECDA 384-bit signature fits in two elements (recall each element is 298 bits). The security of this design follows directly from [18]. We describe the implementation of the bitmap filtering transformation F in more detail in Sect. 4, and the performance optimizations in Sect. 5.

Algorithm 1. Compliance Predicate $\prod(z_{in} = (I_{in}, h_{in}, \sigma_{in}), z_{\mathsf{loc}} = (F, \gamma), z_{out} = (I_{out}, h_{out}, \sigma_{out}), b_{\mathsf{base}})$

```
1: if $b_{\mathsf{base}}$ then                                          ▷ i.e., base case
2:     return $F \in \mathcal{F}$ and $F(I_{in}, \gamma) == I_{out}$ and $h_{in} == H(I_{in})$
3: else
4:     return $F \in \mathcal{F}$ and $F(I_{in}, \gamma) == I_{out}$ and $h_{out} == h_{in}$
5: end if
```

3.2 DCDS from Proof Aggregation

Our proof aggregation DCDS design is shown in Fig. 2(b) and provides similar assurances. Our instantiation of the proof aggregation is however more efficient primarily because of using a more efficient hashing circuit and parallelization as we shall describe in Sect. 5.4 and Sect. 5.5. As with the PCD design, the sensor produces an image I_0, hashes it to $h_0 = H(I_0)$, signs the hash using its private signing key v_s to produce σ_h. It sends the tuple (h_0, σ_h) directly to the boundary node (the verifier), and it sends the image I_0 to the successor filtering node F_0. Each filtering node i in the pipeline uses pp-zk-SNARK prover algorithm to generate a proof π_{i+1} attesting to the following statements:

1. $h_i == H(I_i)$
2. $I_{i+1} == F_i(I_i)$
3. $h_{i+1} == H(I_{i+1})$

Formally, node i computes $P(\mathsf{pk}, (h_i, h_{i+1}), (I_i, I_{i+1})) \rightarrow (\pi_{i+1})$, where the hashes are the public input and the images are the private witness. Node i directly sends the tuple (π_{i+1}, h_{i+1}) to the domain boundary. Notice that both the proof and the hash are very small on the order of hundreds of bytes, incurring little extra communication cost. The last filtering node in the pipeline (node 2 in Fig. 2(b)) additionally sends its output filtered image (I_3) to the domain boundary.

Given the final transformed image I_n (for an n step pipeline) and all the intermediate hashes and proofs $((h_n, \pi_n), \ldots, (h_1, \pi_1), h_0, \sigma_h)$, the boundary node verifies the following conditions:

– $h_n == H(I_n)$, this is performed outside the SNARK
– $V(\mathsf{vk}, (h_i, h_{i+1}), \pi_{i+1}) == 0$, $\forall i \in [0, n-1]$
– $V_S(p_s, h_0, \sigma_h) == 0$, i.e., σ_h is a valid signature of h_0 under the public verification key p_s; this is also performed outside the SNARK

The boundary node passes the filtered image across the boundary if and only if all these conditions are met.

The security of this scheme follows from the following facts. If $H(I_n) == h_n$ and $V(\mathsf{vk}, (h_{n-1}, h_n), \pi_n) == 0$, then I_n must equal $F_n(I_{n-1})$ for some I_{n-1}. Similarly, recursively verifying each of the proofs proves there is some original image I_0 from which I_n is derived according to a sequence of valid filters F_i. If in addition σ_h is a valid signature on h_0, the domain boundary can prove that I_0 is authentic (produced by the sensor with possession of the signing key).

4 Implementation

We implement, optimize, and evaluate bitmap (BMP) image filtering for both the recursive proof composition and proof aggregation DCDS designs. We evaluate the implemented compliance predicate in terms of the total number of image pixels $N = w \times h$, where w and h are the width and height of the image. Our C++ implementation is built on top of libsnark [3].

In general, a BMP file consists of five main parts; a file header, image header, color table, pixel array, and an International Color Consortium (ICC) color profile. The image header is the most complex from a compatibility standpoint due to the varying versions. However, the BITMAPINFOHEADER format introduced in Windows 3.0 is the most commonly used format for compatibility reasons and is the focus of this work. We disregard bitmaps with the optional ICC color profile since these are less common and are only supported under version 5 image headers. And we only consider uncompressed bitmaps with 24 bit color depths, as these are most common in practice.

Bitmap Inspection and Sanitization. We implement a compliance predicate $\prod$ adhering to the National Security Agency's Inspection and Sanitization guide (ISG) for Bitmaps [11]. The ISG provides an analysis on various elements that are contained within the BMP file structure and how they can be a cause for concern for either hiding sensitive data or attempts to exploit a system. We implement a majority of the recommendations for mitigating these threats.

File Header Compliance The file header is a 14-byte structure that stores general information about the BMP. It begins with the magic bytes `0x424D`, and then defines the file size, reserved bytes, and offset address of the pixel data.

Image Header Compliance The BMP Info Header image header type is 40 bytes and contains general information about the size, compression type, number of planes, resolution, and bit count of the BMP. The ISG outlines data *attack* and data *hiding* concerns with BMP image headers. Most attacks are prevented though the zeroing out of unused fields, or ensuring that default values are used. Our compliance predicate ensures the size of the image header is 40 bytes, the colors used value equals 0 (no color table present), the colors important value equals 0 (default value indicating that all colors are required), compression value equals 0 (compression not used), number of planes equals 1 (only supported value for BMP files), and that the width and height fields correspond to the size of the image.

Color Table The Microsoft Developer Network (MSDN) states that the color table is optional in Bitmaps with $\geq$ 8 bit color depths. Additionally, the NSA's inspection and sanitization guide for BMP files recommends removing the color table in BMP files with 24 bit color depths. For this reason, we only consider BMP files without a color table present.

Bitmap Pixel Filtering In addition to filtering the headers of the BMP file, our compliance predicate filters the BMP pixels. We implement a simple compliance predicate (filter) for *redaction* which performs any of the following: identity transformation, blacking out of image pixel regions, and/or cropping the images in a single compliance predicate.[1] The transformation is defined by a $w \times h$ redaction matrix R of boolean values. The constraints over R require that $R_{i,j} \times I_{i,j} = O_{i,j}$ where I is the input pixel matrix, and O is the output pixel matrix. The booleanity of R is enforced by requiring $R_{i,j} \times (1 - R_{i,j}) = 0$. This simple compliance predicate can simultaneously do *cropping*, *black-out boxes*, and *identity* transformations using only $2N$ multiplication gates.

5 Baseline and Optimizations

We implement several optimizations and evaluate their performance against a strong baseline. We show a 20$\times$ speedup in prover time over the state-of-the-art [18], allowing us to filter over large 900×900 images. We describe the baseline and each of the optimizations next.

5.1 PCD Baseline

The baseline is based on the state-of-the-art image authentication from PCD [18]. Given that the source code of [18] is not publicly available, we implemented our BMP filtering predicate within libsnark's [3] PCD implementation which uses BCTV14 [6] pp-zk-SNARK for recursive proof composition over the MNT4 and MNT6 cycles of elliptic curves. Following [18], we also use the subset-sum hash function for hashing field elements, and we implement digital signatures outside the PCD. We verified that the number of constraints in our baseline circuit closely match those reported in [18].

5.2 Switching to Groth16

We switch to the Groth16 proving system [16] from BCTV14 [6], and implement the Groth16 verifier circuit in the PCD. The Groth16 proving system has faster verification and size-optimal proofs for pairing-based arguments. Faster verification and smaller proofs naturally imply less computation in the proof verification

[1] We also added other useful transforms (e.g., downscaling the image by a some factor) which we do not describe here for simplicity but they are part of the codebase.

portion of the prover's circuit as there are fewer input wires and fewer gates. As a concrete comparison, Groth16 proofs consist of only 2 $\mathbb{G}_1$ elements and 1 $\mathbb{G}_2$ element compared to BCTV's proofs of 7 $\mathbb{G}_1$ elements and 1 $\mathbb{G}_2$ element. The smaller Groth16 proofs result in a verification savings of 9 fewer pairings and 4 fewer pairing-product equations used for verifying proofs.

More importantly, the Groth16 prover requires fewer multi-exponentiations compared to BCTV14. Groth16 uses $5m - 2n$ less exponentiations in $\mathbb{G}_1$ and $m - n$ less exponentiations in $\mathbb{G}_2$, where $m \geq n$ represent the number of wires and n the number of multiplication gates respectively. We also use the asymmetric bilinear map construction for efficiency. Asymmetric elliptic curves are more practically and efficiently realizable for higher security levels [12]. Note that the savings in exponentiations mainly apply to dense R1CS statements. In other words, if the majority of R1CS constraints involve wires that carry a 1 or a 0, then the multi-exponentiations become cheap, in which case Groth16 doesn't help reduce the prover cost.[2] This is indeed the case for our prover's circuit since the majority of the constraints are due to *booleanity*, the unpacking of field elements to bits for hashing. However, when combined with our second optimization (next section) which significantly reduces the booleanity constraints, the savings become more noticeable.

5.3 Reducing Booleanity Constraints

The PCD hashing circuit hashes the bit representation of the string $\mathsf{vk}||z$ where vk is the verification key and z is the input message which includes the image in our application. Unpacking of these field elements to bits in order to hash them is very expensive. The overall size of the main PCD circuit C_{pcd} is $|\prod| + s \cdot 89412 + (1+s) \cdot N \cdot 298 + 11925$ gates, where s is the arity (number of incoming messages to each node) and N is the input size (see Appendix A and [6] § 5 for details). The term $11920 + (1+s) \cdot N \cdot 298$ costs around 10 million gates for even a small 128×128 image where $N = 16384$ pixels or field elements each is 298 bits (the arity $s = 1$).

The binary representation of the verification key is 11920 bits (gates). On the other hand, the binary representation of the inputs z costs $(1+s) \cdot N \cdot 298$ gates since each element of the input is represented with $\lceil \log r_4 \rceil = 298$ bits (gates). However, since we know that each element of the input message (a raw pixel) is represented with fewer than 32 bits, we can truncate the input before hashing it. We unpack each pixel f to its binary representation and enforce $f = \sum_{i=1}^{32} b_i \cdot 2^{i-1}$ for each bit b_i of f. Implicit in this constraint is that the 266 most significant bits of f must equal 0 if it holds, thereby avoiding the booleanity checks on those values. This optimization reduces the booleanity gates by a factor of $298/32 = 9.3\times$. For a 128×128 image, the resulting circuit is reduced from 10 million to about 1 million gates. It also reduces the number of variables (wires in the circuit) and accordingly the proving key size by about $6\times$.

[2] In this case FFTs dominate the prover's cost, and the cost of FFTs in both BCTV14 and Groth16 are the same.

Table 1(b) shows the significant improvements resulting from these first two optimizations. Specifically, at the 250×250 image size, we see around $9\times$ reduction in number of constraints in the circuit (from 37.8M to 4.3M), $5\times$ reduction in prover time, and $6.6\times$ reduction in peak memory utilization at the prover.

5.4 Algebraic Hash Functions

In both the PCD and proof aggregation settings, the vast majority of the compliance predicate's constraints are due to the hashing of input and output images. Even with the booleanity optimization, the booleanity constraints continue to dominate the cost of our circuit. Traditional hashing algorithms such as SHA are complex and not well-suited for SNARK applications since they require converting $\mathbb{F}_p$ elements to bits, incurring a non-trivial $\lceil \log(p) \rceil$ multiplication gates per field element cost. This roughly amounts to 1 constraint per bit per field element and dominates our circuit's overall cost. For this reason, recent work has proposed algebraic hash functions whose domain is $\mathbb{F}_p$ [15].

We instantiate the Poseidon hash function [15] over $\mathbb{F}_p$ and analyze the performance benefits. First, we observe that Poseidon achieves on the order of 0.3 constraints per bit, compared to the subset-sum's cost of 1 constraint per bit (due to the unpacking of bits from $\mathbb{F}_p^m$). However, our booleanity optimization from earlier reduced the cost from 1 constraint per bit to around 0.1 constraints per bit. We observe that rather than truncating the field elements for the algebraic hashing gagdet, we can instead pack multiple pixels into a single field element. For the BN254 curve, we are able to squeeze 7 pixels into a single 254 bit field element. We use a pixel packing gadget that ensures for a field element $X \in \mathbb{F}_p$ and a set of pixel values f, $\sum_{i=0}^{6} 2^{32i} \cdot f[i] == X$.

We implement the Poseidon hashing for the proof aggregation setting only, since integrating it into PCD is non trivial and beyond the scope of this paper. The results are shown in Table 1(c), showing an additional $3\times$ reduction in number of constraints in the circuit translating to $5\times$ speedups in prover time at the 250×250 image size. Note that the difference in performance between Table 1(c) and Table 1(b) is primarily because of the Poseidon hashing, i.e., we expect the two sets of results to be close once Poseidon is integrated into PCD. This is because the number of constraints for PCD's unique verification circuit (the V component of C_{pcd}) are very small relative to the number of constraints due to the compliance predicate (the $\prod$ component of C_{pcd}).

5.5 Reducing Pipeline Latency

The end-to-end latency of the DCDS filtering pipeline is defined as the time from image publication at the sensor until the filtered content crosses the boundary (or fails). As shown in Fig. 2, this latency depends on the depth of the pipeline, the number of filtering steps in our example. In a naive instantiation of the DCDS pipelines of Fig. 2 whereby each filter generates the proof and forwards the output(s) to the next filter in the pipeline, the end-to-end latency t_k for a pipeline of depth k is $O(kt_{prover})$, where t_{prover} is the prover time (the time each filter needs in order to produce a proof over its computation). For example, Table 1(b) (Table 1(c)) show the prover time to be 62 s (15 s) for a 250×250 image. Multiplying these numbers by the depth of the pipeline can get expensive.

We make t_k independent of k in the proof aggregation setting of Fig. 2(b) i.e., $t_k \approx O(t_{prover})$. When a filter node F_i receives its input image I_i, it *natively* executes the equivalent traditional filtering software to produce the filtered output image I_{i+1}. Let t_{F_i} denote this native execution latency, and note that this t_{F_i} is orders of magnitude faster than t_{prover}. Node F_i forwards I_{i+1} to the next filtering node in the pipeline. In parallel to the native execution, Node F_i runs the expensive prover algorithm on the input image and forwards (h_{i+1}, π_{i+1}) to the boundary node as described in Sect. 3.2. As soon as F_{i+1} receives I_{i+1} from F_i, it can immediately begin performing its local computation, rather than waiting on F_i to produce (h_i, π_i). For a pipeline of depth k, it can be shown that the end-to-end latency t_k is reduced from $k(t_{prover} + t_{comm} + t_{verifier})$ to $(\sum_{i=0}^{k-1} t_{F_i}) + t_{prover} + k(t_{comm} + t_{verifier})$, where t_{comm} is the one-hop communication latency.

A similar approach can be used to reduce the latency of the PCD setting (Fig. 2(a)). At first glance, it appears that one cannot parallelize the distributed computation since each sequential hop requires a proof of correctness from the previous DCDS node. However, we observe that the majority of the prover's computation is independent of the previous proof, and can accordingly start in parallel. For example, a monolithic prover can split the circuit into two sections (a) and (b) such that (a) corresponds to wire values independent of the previous proof π_{i-1} and (b) corresponds to wire values dependent on π_{i-1}. The prover then applies techniques from [20] to compute a proof π_i by first computing the values associated with (a), and later computing the values associated with (b) when π_{i-1} arrives. It can be shown that the majority of the prover's computation involves wire values *independent* of the previous proof π_{i-1}, and therefore the majority of the distributed PCD computation can be performed in parallel by forwarding native execution output. We leave this PCD Implementation for future work.

6 Performance Evaluation

We evaluate the performance (compute and memory costs) of the designs for a single hop of the filtering pipeline in terms of the prover time (time for each filter F_i to produce a proof on its local computation) and peak memory, and verifier time (time for the boundary node to verify a single proof π_{i+1}), for different image sizes. The prover's memory footprint includes loading the proving key $\prod$ corresponding to the compliance predicate pk, where the size of the proving key in MB is denoted by $|\mathsf{pk}|$, and $|\prod|$ is the size of the compliance predicate in terms of number of R1CS constraints. Table 1 shows the performance results of three

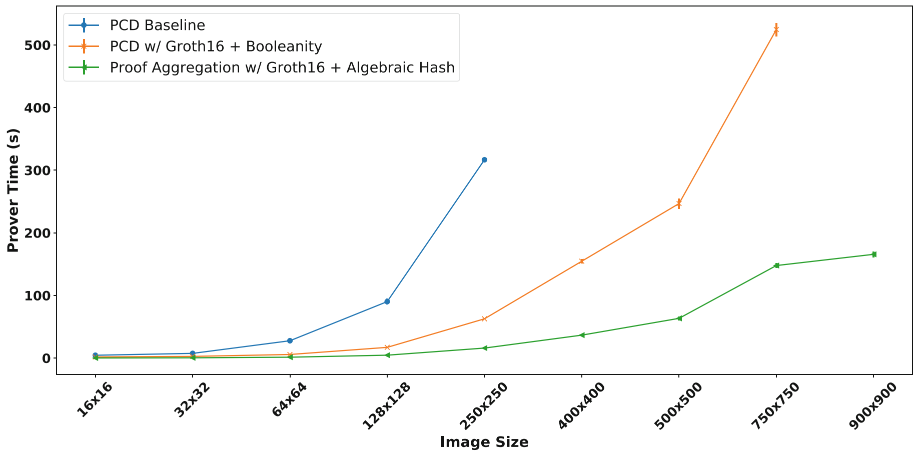


Fig. 3. Comparison of baseline, optimized PCD, and aggregation designs in terms of prover's time (seconds) for different image sizes. The x-axis is quadratic.

different systems on an AWS c5.18xlarge instance with 72 cores and 144 GB RAM. Table 1(a) shows the results for the baseline PCD system of Sect. 5.1. The baseline prover algorithm runs out of memory for images larger than 250×250 pixels. At the 250×250 image size, the filtering predicate has more than 37M constraints and takes the prover more than 5 min and around 100 GB of memory to generate a proof! The proof size $|\pi|$ is 2988 bits.

Table 1(b) shows enhanced performance of the baseline PCD system when incorporating the Groth16 and Booleanity optimizations described in Sect. 5. The optimizations result in $5\times$ speedup in prover time at the 250×250 image size, and the reduction in prover memory allow increasing the image size by $9\times$ before running out of memory. The proof size is reduced to 1493 bits.

Table 1(c) shows the performance of the proof aggregation system with comparable assurances, including Groth16 and algebraic hash optimizations from Sect. 5. At the 250×250 image size, the optimized proof aggregation system delivers $20\times$ speedups in prover time. Additionally, proof verification time is constant (independent of the input image size) since constant-size hashes make up the public input of the circuit. In this proof aggregation design however, the aggregator must verify k proofs for a depth k pipeline each on the order of 5 ms as shown in Table 1(c). The $30\times$ reduction in number of R1CS constraints further enables scaling to large 900×900 image sizes, a $13\times$ larger scale than baseline. The proof size is 1019 bits.

Finally, Fig. 3 compares the three proof systems in terms of prover time showing the relative speedups attained and the resulting increase in image scale. Note that the x-axis in the figure is quadratic, and the scaling is linear in pixels.

Table 1. Performance of the DCDS proof system for a single hop computation on an AWS c5.18xlarge instance with 72 cores and 144 GB RAM. $|\prod|$ is the number of R1CS constraints in the DCDS circuit which contains the predicate. Generator, prover, verifier times are in seconds. $|\pi|$ is the proof size in bits, and Mem is prover's memory in GB. The table compares three systems: (a) unoptimized PCD based proof system, (b) Optimized PCD based proof system, and (c) optimized proof aggregation based proof system. The 250 × 250 image size results are highlighted for comparison.

(a) PCD Baseline							
Image Size	$|\prod|$	$|\mathsf{pk}|$ **(MB)**	***Generator***	***Prover***	***Verifier***	**Mem**	$|\pi|$ **(bits)**
64×64	2,780,396	750.0	80.32	27.67	0.062	12.37	2988
128×128	10,153,196	2,740.8	281.21	90.18	0.096	30.79	2988
250×250	37,822,796	10,351.6	686.77	316.70	0.220	99.48	2988
(b) PCD w/ Groth16 + Booleanity optimizations							
64×64	353,783	82.7	11.35	5.73	0.025	6.11	1493
128×128	1,189,367	289.7	37.28	17.12	0.043	8.24	1493
250×250	4,325,255	1,073.6	119.01	62.72	0.107	15.66	1493
400×400	10,928,055	2,782.4	246.59	154.66	0.246	31.88	1493
500×500	17,109,255	4,274.8	379.63	246.81	0.377	46.56	1493
750×750	38,325,255	9,698.1	706.07	524.71	0.822	98.74	1493
(c) Proof Aggregation w/ Groth16 + Algebraic hash optimizations							
64×64	85,227	14.4	2.29	1.41	0.005	5.78	1019
128×128	338,655	57.3	7.98	4.75	0.005	7.41	1019
250×250	1,289,571	212.5	18.96	15.98	0.005	14.02	1019
400×400	3,301,043	569.5	50.40	36.63	0.005	26.87	1019
500×500	5,157,987	850.0	69.23	63.45	0.005	39.17	1019
750×750	11,604,043	1,936.3	182.29	147.78	0.005	81.59	1019
900×900	16,709,987	2,747.6	248.07	165.70	0.005	114.89	1019

7 Related and Future Work

To our knowledge, this is the first design and implementation of a CDS filtering pipeline based on verifiable computation. Furthermore, we are not aware of any work on decentralizing the CDS architecture across several potentially untrusted devices. Our work is inspired by PhotoProof [18], which describes an image authentication proof-of-concept library using the proof-carrying data scheme of [6]. PhotoProof demonstrated the feasibility of distributed image filtering, however, PhotoProof's ability to scale to larger images was largely limited due to the unavailability of modern algebraic hashing techniques [1,15] and optimized proof systems [16].

In terms of future work, we identify the following three thrusts:

Expressiveness, Performance, and Security. The ability to efficiently filter more complex data types is critical for CDS adoption and for mission enablement. In addition to bitmap filtering, we implemented filters for redacting text documents that use the same principles as the BMP filters discussed here (treating characters in the document like pixels), and we plan to continue enhancing the expressiveness and efficiency of these predicates. Additionally, we plan to experiment with newer constructions of PCD based on accumulation schemes [8], which realize recursive proof composition without the sublinear-time verification requirement. These constructions may permit higher levels of security avoiding for example the limitations inherent to PCD constructions from cycles of elliptic curves [6], while additionally providing post-quantum security properties.

Confidentiality Protection The present DCDS design is primarily concerned with protecting the integrity of the filtering pipeline even when computed on low assurance devices. A filtering node in the current design sees the plaintext input images as well as the witness, both of which may need to be confidential. We plan to extend the design such that the filtering nodes are able to perform a *verifiable computation* over *encrypted data* keeping both the input and the witness private from the filtering node. This problem is akin to verifiable and private delegation of computation [13]. However, these protocols require ideas from fully-homomorphic encryption (FHE) and verifiable computations simultaneously and are currently expensive. We are experimenting with simpler partially homomorphic schemes that are efficient and suffice for some of the desired computations.

Hardware Acceleration To support filtering of information flows that require low latency, the prover time must be further reduced. Consider for example the pp-zk-SNARK prover's optimized algorithm is defined [5] (see § 4.3 and Fig. 10-b in [5]). There are two main operations the prover runs: computing the coefficients h of a polynomial $H(z)$, and computing the proof π using 8 large multi-scalar multiplications of the form $\alpha_1 P_1 + \ldots + \alpha_n P_n$ where P_i are elements of group G_1 (or G_2) and α_i are scalars. Both of these sets of operations can directly benefit from hardware acceleration. We will investigate using GPU or FPGA implementations of FFTs over big integers and multi-scalar multiplications. We expect this may lead to up to two orders of magnitude speedups for large circuits.

8 Conclusion

This paper presents a first step towards building decentralized and provably secure cross domain solutions, allowing secure composition of CDS components (such as content filters) running on untrusted edge devices. A decentralized CDS capability complements today's enterprise and tactical CDS enabling secure cross-domain information sharing when access to the enterprise or single-device tactical CDSs is not available, such as with the disconnected, intermittent, and limited tactical edge. We show that instantiating such a decentralized capability can be made practical, for the simple image filtering predicates considered in this paper. We expect the active research efforts for enhancing the practicality of verifiable computations will render this paradigm increasingly more practical in the future.

A From zk-SNARK to PCD

A PCD system (*Generator*, *Prover*, *Verifier*) is constructed in [7] using *recursive composition of pp-zk-SNARK* (G, P, V) *proofs*. The main idea behind recursive proof composition is that the new proof system has to prove two things now at each node: (1) *the proof of the previous computation step is valid* **and** (2) *the node performed a valid local computation*. In other words, the pp-zk-SNARK is used to prove that the input proof $\boldsymbol{\pi_{in}}$ attests to the compliance of $\boldsymbol{z_{in}}$, **and** the output z is $\prod$-compliant given $(\boldsymbol{z_{in}}, z_{\mathsf{loc}})$. This effectively allows recursion so that the *history* can now be discarded at each step, and hence enables compliance predicate verification only by looking at the proof and data from the last step.

More concretely, in order to construct the recursive PCD proof system, the PCD circuit C_{pcd} must encode the pp-zk-SNARK verification algorithm V in addition to the local computation corresponding to the compliance predicate $\prod$ i.e., one must construct the $\mathbb{F}$-arithmetic circuit C_V corresponding to V as a sub-circuit of C_{pcd}.

A known efficient pp-zk-SNARK verification function uses pairings on elliptic curves [6]. Since, the verification function, the circuit C_V, operates over the base field $\mathbb{F}_q$ of the curve rather than over $\mathbb{F}_r$ over which the NP statement is defined,[3] realizing C_{pcd} in practice is challenging (see [6] for details). For the sake of this discussion, we just mention that C_{pcd} involves a lot more than the local computation. There are two separate PCD circuits, each one on a different elliptic curve, such that the two curves are on a *cycle*. The main PCD circuit C_{pcd} does three things:

1. implements a collision-free hash function that verifies the output hash of vk, z is valid, which involves circuits for bit conversion because the hash function operates over bit strings (this step is required in order to bypass the circular dependency between the two proof systems generated from the two curves on a cycle)
2. verifies the local predicate $\prod(z, \boldsymbol{z_{in}}, z_{\mathsf{loc}}, b_{\mathsf{base}})$
3. recursively verifies $C_V(\mathsf{vk}, \boldsymbol{z_{in}}, \boldsymbol{\pi_{in}})$ for each pair which also involves circuits for bit conversion

Circuit Size. The overall size of the main PCD circuit C_{pcd} is $|\prod| + s \cdot 89412 + (1+s) \cdot n \cdot 298 + 11925$ gates, where s is the arity (number of incoming messages to each node) and n is the input size (see [6] §5 for details; there is also an auxiliary PCD circuit we don't show here as it has constant cost.). This shows the (additive) dependence of the prover cost on $|\prod|$. Besides the predicate, a main contributor to cost is the booleanity checks which requires expanding into their bit representations each of the input and output messages ($(1+s) \cdot n \cdot 298$ gates), where $\mathbb{F}_r$ is a prime field of 298 bits. For a large input such as a 128×128 image i.e., $n = 16384$ field elements each 298 bits, this term can be large requiring

[3] Here q is the size of the base field over which the curve is defined, and r is the order of the group (number of points on the curve).

around 10 million gates even for $s = 1$, far exceeding the cost of the predicate $\prod$. This $\lceil \log r_\alpha \rceil = 298$ blow up factor seems to be inherent to the construction because the collision-resistant hash function operates on binary string inputs (see [6] §4.1), and expanding a field element x to its bit representation requires $\lceil \log r_\alpha \rceil$ constraints to verify $\sum_i b_i 2^i = x$, where b_i is the bit at index i in x's binary representation.

Prover Key Size and Memory. The prover key is made of a large set of group elements. Here, we relate the number of group elements in the proving key to the input and circuit dimensions to understand the effect of circuit complexity on performance. The number of elements in the key depends on the Quadratic Arithmetic Program (QAP) instance [14], which is derived from the Rank-1 Constraint System (R1CS) through an efficient reduction (see [4] Appendix E). Briefly, a R1CS constraint system is expressed as $A \cdot s \bigodot B \cdot s = C \cdot s$, where s is a vector of $m + 1$ variables (input, intermediate, and output variables) corresponding to the m wires in the arithmetic circuit (an additional special variable, one, is used resulting in $m+1$ variables). And A, B, and C are matrices of dimension $l \times m + 1$ for a system with l constraints corresponding to the l gates of the circuit (each row corresponds to a constraint).

A R1CS constraint system is reduced to a QAP instance with the same number of $m + 1$ variables and whose degree is $d(l)$, where $d(l)$ is some value larger than l selected for an evaluation domain to optimize computations of Lagrange polynomials and FFT/iFFT. The QAP instance is similarly represented with three sets of polynomials A', B', and C' each containing $m + 1$ polynomials each of which is degree $d(l)$. In summary, the resulting proving key contains at most $6m + d(l) + 13$ elements from group $\mathbb{G}_1$ and $m + 4$ elements from group $\mathbb{G}_2$. Reducing m, the number of wires in the circuit, significantly affects performance (key size, memory, generator time, and prover time). The proof always has 7 $\mathbb{G}_1$ elements and 1 $\mathbb{G}_2$ element [5].

References

1. Aly, A., Ashur, T., Ben-Sasson, E., Dhooghe, S., Szepieniec, A.: Design of symmetric-key primitives for advanced cryptographic protocols. IACR Trans. Symmetric Cryptol. **2020**, 1–45 (2020)
2. Australian Cyber Security Centre: Fundamentals of cross domain solutions, June 2020. https://www.cyber.gov.au/acsc/view-all-content/publications/fundamentals-cross-domain-solutions, Accessed 10 Apr 2020
3. Ben-Sasson, E., Chiesa, A., Genkin, D., Kfir, S., Tromer, E., Virza, M.: libsnark, 2014. https://github.com/scipr-lab/libsnark
4. Ben-Sasson, E., Chiesa, A., Genkin, D., Tromer, E., Virza, M.: SNARKs for C: verifying program executions succinctly and in zero knowledge. In: Canetti, R., Garay, J.A. (eds.) CRYPTO 2013. LNCS, vol. 8043, pp. 90–108. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40084-1_6
5. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Succinct non-interactive zero knowledge for a von Neumann architecture. In: USENIX Security 2014, pp. 781–796 (2014)

6. Ben-Sasson, E., Chiesa, A., Tromer, E., Virza, M.: Scalable zero knowledge via cycles of elliptic curves. Algorithmica **79**(4), 1102–1160 (2017)
7. Bitansky, N., Canetti, R., Chiesa, A., Tromer, E.: Recursive composition and bootstrapping for snarks and proof-carrying data. In: Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing, pp. 111–120 (2013)
8. Bünz, B., Chiesa, A., Mishra, P., Spooner, N.: Proof-carrying data from accumulation schemes. IACR Cryptol. ePrint Arch. **2020**, 499 (2020)
9. Chiesa, A., Tromer, E.: Proof-carrying data and hearsay arguments from signature cards. In: ICS, vol. 10, pp. 310–331 (2010)
10. Chiesa, A., Tromer, E.: Proof-carrying data: secure computation on untrusted platforms (high-level description). Next Wave Natl. Secur. Agency Rev. Emerging Technol. **19**(2), 40–46 (2012)
11. Cross Domain Products and Technology Branch of the Information Assurance Directorate: Inspection and Sanitization Guidance for Bitmap File Format. Technical report, Version 1.0, National Security Agency, January 2012
12. Galbraith, S., Paterson, K., Smart, N.: Pairings for cryptographers. Cryptology ePrint Archive, Report 2006/165 (2006). https://eprint.iacr.org/
13. Gennaro, R., Gentry, C., Parno, B.: Non-interactive verifiable computing: outsourcing computation to untrusted workers. In: Rabin, T. (ed.) CRYPTO 2010. LNCS, vol. 6223, pp. 465–482. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-14623-7_25
14. Gennaro, R., Gentry, C., Parno, B., Raykova, M.: Quadratic span programs and succinct NIZKs without PCPs. In: Johansson, T., Nguyen, P.Q. (eds.) EUROCRYPT 2013. LNCS, vol. 7881, pp. 626–645. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-38348-9_37
15. Grassi, L., Khovratovich, D., Rechberger, C., Roy, A., Schofnegger, M.: Poseidon: a new hash function for zero-knowledge proof systems. In: Proceedings of the 30th USENIX Security Symposium. USENIX Association (2020)
16. Groth, J.: On the size of pairing-based non-interactive arguments. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 305–326. Springer, Heidelberg (2016). https://doi.org/10.1007/978-3-662-49896-5_11
17. National Cross Domain Strategy and Management Office: Cross Domain Solution (CDS) Design and Implementation Requirements: 2020 Raise the Bar (RTB) Baseline Release. Technical report, NCDSMO-R-00008-003_00, National Security Agency, December 2020
18. Naveh, A., Tromer, E.: Photoproof: cryptographic image authentication for any set of permissible transformations. In: 2016 IEEE Symposium on Security and Privacy (SP), pp. 255–271. IEEE (2016)
19. Smith, S.: Shedding light on cross domain solutions (2015). https://www.sans.org/white-papers/36492/. Accessed 10 Apr 2020
20. Wu, H., Zheng, W., Chiesa, A., Popa, R.A., Stoica, I.: DIZK: a distributed zero knowledge proof system. In: USENIX Security 2018, pp. 675–692 (2018)