Algorithms for Constructing Anonymizing Arrays

Erin Lanus[1(✉)] and Charles J. Colbourn[2]

[1] Virginia Tech, Arlington, VA 22203, USA
lanus@vt.edu
[2] Arizona State University, Tempe, AZ 85281, USA
charles.colbourn@asu.edu

Abstract. Attribute-based methods are inherently identity-less as authorization decisions are made in terms of attributes possessed by the subject rather than identity. However, anonymity against the system is not guaranteed when attribute distribution allows for the composition of a policy that few subjects can satisfy. An anonymizing array ensures that any assignment of values to t attributes that appears in the array appears at least r times. When an anonymizing array is used for subjects registered to a system and policies contain conjunctions of at most t attributes, the system cannot identify the subject using the policy to to gain authorization with greater than $\frac{1}{r}$ probability. Anonymizing arrays are similar to covering arrays with higher coverage and constraints, but have an additional desired property, homogeneity, due to their application domain. In this paper, we develop constructions for anonymizing arrays and propose a post-optimization mechanism to reduce homogeneity.

Keywords: Combinatorial array · Construction algorithms · Anonymous authorization · Attribute-based methods

1 Introduction

In attribute-based systems used for access control, such as Attribute-Based Access Control and Ciphertext-Policy Attribute-Based Encryption (CP-ABE), decisions are made on the basis of attributes, or characteristics of a subject expressed as name-value pairs [1,5]. A feature of these systems is that they can achieve anonymous access control, granting access to authorized subjects and denying access to unauthorized subjects without knowledge of the subject's identity. This is not a guarantee that the identity cannot be deduced. CP-ABE encrypts a ciphertext in a policy, and decryption is performed by a private

Research of EL was supported by a National Physical Science Consortium Fellowship. Research of CJC was supported in part by the National Science Foundation under Grant No. 1421058 and Grant No. 1813729.

L. Gąsieniec et al. (Eds.): IWOCA 2020, LNCS 12126, pp. 382–394, 2020.
https://doi.org/10.1007/978-3-030-48966-3_29

key containing attributes satisfying the policy. CP-ABE is proposed to mediate authenticated key exchange with an anonymous mode [9]. Suppose a service broadcasts a session key encrypted by a policy. A subject whose private key contains attributes satisfying the policy decrypts the message, obtains the key, and begins communicating with the service via the session key. The service knows that the subject communicating with it is authorized based on possession of the required attributes to obtain the key. The authors claim that the service cannot uniquely identify the subject. All subjects must register with the service to receive a private key, and thus the service knows all attributes of the subjects in the system. If a policy can be composed so that only one subject's attributes satisfy the policy and this policy is used to encrypt the session key, the service knows the identity of the subject using this session key. "Anonymous ABE" uses hidden credentials which can be used to retrieve a session key anonymously, but the receiver anonymity is based on "plausible deniability" due to the fact that anyone can request the message, not just the intended recipient [6]. Plausible deniability fails if the message is decrypted to gain a session key to obtain authorization, proving that a subject with the correct credentials decrypted the message.

The contribution of this work is to achieve a guaranteed degree of anonymity by requiring that certain properties of attribute distribution hold given a maximum credential size. Policies can be considered disjunctions of conjunctions of attribute values with the most restrictive policy being a single conjunction of many attribute values. Let t be the largest number of attributes in a single conjunction. An *anonymizing array* ensures that any assignment of values to t attributes that appears in the array appears at least r times [7]. When an anonymizing array is used for subjects registered to a system and policies contain conjunctions of at most t attributes, the system cannot identify the subject using the policy for authorization with greater than $\frac{1}{r}$ probability.

An *access profile* is an assignment of values to attributes. When attributes are assigned to access profiles for the purpose of anonymous authorization, as in key distribution, rather than existing as real-world attributes of subjects, an anonymizing array is built from scratch. When the set of subject attributes registered to a system is fixed, an anonymizing array determines the largest conjunction that can be used while achieving the anonymity guarantee r or, equivalently, the guarantee achievable for the largest conjunction. When subject attributes are immutable but the set of access profiles can be appended, anonymizing arrays provide a mechanism to provide higher anonymity guarantees. Constructions for anonymizing arrays must account for constraints on attributes and, due to the security application, must not rely randomness in order to achieve the guarantee with a high probability. The rest of the paper is organized as follows. Definitions and relationship to covering arrays are in Sect. 2. Construction algorithms are in Sect. 3. Results are in Sect. 4, and conclusions are in Sect. 5.

2 Anonymizing Arrays

2.1 Definitions

Consider an array with N rows and k columns and each column i for $1 \leq i \leq k$ has entries from a set of v_i symbols. The rows of the array are access profiles, columns are attributes, and symbols in a column are the values for the attribute. To express the parameters of the array, write $\mathsf{AA}(N; r, t, k, (v_1, \ldots, v_k))$ or use exponential notation v_i^j when j columns share the same number of symbols v_i. Write $\mathsf{AA}(N; r, t, k, v)$ when the number is the same for all columns. Such an array is (r, t)-*anonymous* if, when choosing an $N \times t$ subarray, $1 \leq t \leq k$, each row that appears is repeated at least r times. A *credential* is a tuple of attribute-value pairs presented for an authorization decision. The *maximum credential size* is t and r is the *anonymity guarantee*. Given an $N \times k$ array $\mathbf{A}$, an $N' \times k$ array $\mathbf{A}'$ is (r, t)-*anonymizing* with respect to $\mathbf{A}$ if $\mathbf{A} \subseteq \mathbf{A}'$ and $\mathbf{A}'$ is (r, t)-anonymous. Interesting cases require $r > 1$ and $v_i > 1$ for all v_i. An access profile may represent a subject or it may be *padding*, a row added to reach an anonymity guarantee. Access profiles need not be unique. The rows or columns of the array can be shuffled to obtain an equivalent array on the same parameters.

Hard constraints are credentials that cannot appear, while *soft constraints* are credentials that need not appear, but are not illegal. That is, hard constraints and non-appearing soft constraints must appear 0 times, while soft constraints that appear and all unconstrained credentials must appear at least r times. Hard constraints may give rise to *implicit hard constraints* that cause there to be no feasible solution. Constraints must be considered when appending padding rows to an anonymizing array to reach the anonymity guarantee r.

Anonymizing arrays containing groups of highly similar access profiles may lead to affiliating subjects with and tracking subjects by their groups; [7] develops the following metrics to detect this similarity. *Local homogeneity* describes how often an access profile appears in small groups of similar access profiles, and *global homogeneity* is the average local homogeneity. The *neighborhood* of a credential is the set of access profiles possessing the credential. The *closeness* of a pair of access profiles is a sum of their *weight* over all credentials, and the weight of a pair of access profiles on a credential is inversely proportionate to the size of the neighborhood of the credential if the access profiles are in the neighborhood. Let $\mathcal{U}$ be a set of N access profiles and let $\mathcal{C}$ be the set of credentials. Define the neighborhood of a credential $c \in \mathcal{C}$ as $\rho(c) = \{u_i : u_i \text{ possesses } c, u_i \in \mathcal{U}\}$ and

$$weight(u_i, u_j, c) = \begin{cases} \frac{1}{|\rho(c)|} & \iff \{u_i, u_j\} \subseteq \rho(c) \\ 0 & \text{otherwise} \end{cases}$$

$$closeness(u_i, u_j) = \sum_{c \in \mathcal{C}} weight(u_i, u_j, c)$$

$$neighbors(u_i) = \{\bigcup_{c \in \mathcal{C}} \rho(c) : u_i \in \rho(c)\}$$

$$homogeneity(u_i) = \frac{1}{|neighbors(u_i)|} \sum_{u_j \in \mathcal{U}, u_j \neq u_i} closeness(u_i, u_j)$$

2.2 Relationship to Covering Arrays

A *covering array* denoted $\mathsf{CA}(N; t, k, v)$ is an $N \times k$ array on v symbols such that in every $N \times t$ subarray each of the v^t combinations of symbols, called *interactions*, appears as a row. When different columns can have different numbers of symbols, it is a *mixed-level covering array* $\mathsf{MCA}(N; t, k, (v_1, \ldots, v_k))$. In rare cases when higher coverage is needed, interactions may be required to appear at least $\lambda > 1$ times. When not specified, $\lambda = 1$ is implied. Anonymizing arrays are similar to covering arrays with constraints and higher coverage. The primary difference due to application is in the desired homogeneity property, but also in how constraints are treated. For covering arrays, the norm is to define the interactions that must not appear (hard constraints), then to define the interactions that might appear (soft constraints, possibly further divided into "don't care" and "avoid"), and then to derive the interactions that must appear [3]. For anonymizing arrays, the access profiles provided define the unconstrained credentials. The system specification defines the hard constraints, and the soft constraints are defined to be the remaining credentials that are in neither set. Given an anonymizing array without a defined set of constraints, it may be impossible to distinguish the soft and hard constraints from the set of non-appearing credentials. The same difficulty arises distinguishing the soft constraints from the unconstrained credentials. Care must be taken when converting between covering arrays and anonymizing arrays that constraints are categorized correctly. Many construction algorithms exist for building covering arrays, though few explicitly include constraint handling or higher coverage requirements. The following non-exhaustive list of relationships elucidate how to use covering array constructions to build anonymizing arrays.

Any $\mathsf{MCA}_\lambda(t, k, (v_1, \ldots, v_k))$ with hard constraint set $\mathcal{H}$ is also an $\mathsf{AA}(\lambda, t, k, (v_1, \ldots, v_k))$ with hard constraint set $\mathcal{H}$ and all other credentials appearing. Every t-way interaction that appears in the covering array λ times is a credential that appears λ times in the corresponding anonymizing array. The interactions in $\mathcal{H}$ never appear in the covering array so they never appear in the anonymizing array. In the context of covering arrays, higher λ does not force a "don't care" or an "avoid" interaction to appear λ times if it appears once. Then soft constraints must not be present in a covering array used as an anonymizing array. There must also exist a mapping of soft constraints in the anonymizing array onto either hard constraints in the covering array if they do not appear or onto unconstrained interactions if they do.

If an $\mathsf{MCA}(t, k, (v_1, \ldots, v_k))$ with hard constraint set $\mathcal{H}$ and soft constraint set $\mathcal{S}$ exists, then an $\mathsf{AA}(r, t, k, (v_1, \ldots, v_k))$ with $\mathcal{H}$ and $\mathcal{S}$ exists. Copy the covering array vertically r times. No interaction of $\mathcal{H}$ appears in the covering array, so none of these credentials appear in the anonymizing array. Any interaction of $\mathcal{S}$ that appears in the covering array at least once appears in the anonymizing array

at least r times, and the rest never appear. Unconstrained credentials appear at least once in the covering array and at least r times in the anonymizing array.

An $\mathsf{MCA}(t, k, (v_1, \ldots, v_k)), v = \min_{i=1}^{k}(v_i)$ with no constrained interactions is an $\mathsf{AA}(v, t-1, k, (v_1, \ldots, v_k))$ with no constrained credentials. In the mixed-level covering array without constraints, a $(t-1)$-way interaction appears at least v_i times, once with each of the v_i symbols in the t-th column of the t-way interaction including those $t-1$ columns. Every $(t-1)$-way interaction appears at least v times, v the minimum v_i. This is an anonymizing array for $r = v$.

If there exists a covering array $\mathsf{CA}(t, k, v)$ with a set of hard constraints $\{(c_1, \sigma_1), \ldots, (c_{t-1}, \sigma_{t-1}), (c_x, \sigma_y)\}$ for each column symbol pair (c_x, σ_y) with column $c_x \in \mathcal{K} \setminus \{c_1, \ldots, c_{t-1}\}$ and $\sigma_y \in \Sigma_x$, the symbol set of c_x, then there is an anonymizing array $\mathsf{AA}(v, t-1, k, v)$ with $\{(c_1, \sigma_1), \ldots, (c_{t-1}, \sigma_{t-1})\}$ as a hard constraint. To guarantee that the constrained credential with $t-1$ attributes never appears in the anonymizing array, it must be the case that no t-way interactions of which it is a subset appeared in the covering array. The coverage for all unconstrained credentials has already been shown. To extend this to soft constraints, there must be a mapping of soft constraints in the anonymizing array to either unconstrained interactions or hard constraints in the covering array.

Given an array $\mathbf{A}$ that is (r, t)-anonymous and not $(r+1, t)$-anonymous, for every $t \leq t' \leq k$ for which $\mathbf{A}$ is (r', t')-anonymous, it must be the case that $r' \leq r$. Pick the credential c that appears the fewest number of times in $\mathbf{A}$ and let r be the number of times c appears. $\mathbf{A}$ is (r, t)-anonymous by definition and is not $(r+1, t)$-anonymous. Choose any credential c' that contains c. The rows in which c' appears must be a subset of the rows in which c appeared. Then for $t' \geq t$, if $\mathbf{A}$ is (r', t')-anonymous, then $r' \leq r$. Similarly, an array that is (r, t)-anonymous is (r, t')-anonymous for $t' < t$. Any credential, c, of size t appears in at least r rows. Any t'-subset of c appears in at least these rows.

3 Construction Algorithms

3.1 Moser-Tardos-Style Column Resampling Algorithm

Algorithm 1 is a Moser-Tardos-style column resampling algorithm (MTCR) [8]. A *bad event* is either a violation of a hard constraint or lack of necessary coverage on unconstrained or soft constraints. A candidate is checked systematically until either no bad events are found or an iteration limit is reached. If any bad event is found, all involved columns are resampled. If $\mathcal{T}$ is the set of $\binom{k}{t}$ t-subsets of columns and there are $\sum_{T \in \mathcal{T}} \prod_{i \in T} v_i$ possible credentials, the position of T in colexicographic ordering of the sets is the *rank*. Estimating the number of rows is not obvious, so rows are added until coverage is met or an iteration limit is reached. When provided a set of rows, MTCR adds padding to meet the guarantee and forbids resampling of initial rows. When building from scratch, the candidate starts with no rows or an initial number of randomly populated rows is computed as r times the maximum number of non-constrained credentials of any rank. Adding rows too often may produce more rows than needed, while the iteration limit may be reached when adding conservatively. Too few rows

Algorithm 1: Moser-Tardos-style Column Resampling (MTCR)

```
input  : A, r, t, k, (v_1, ..., v_k), and a set of constraints
output: A or ∅
begin
    while iterations < limit do
        foreach rank while no bad event do
            Check all credentials in rank
            if coverage bad event then
                Increment number of resamplings
                if resamplings > rank * threshold then
                    Add a row to A and reset resamplings
        if no bad event then
            return A
        else
            Resample all columns of rank in A
    return ∅
```

can contribute to lack of r coverage, but not to presence of a constraint, as more rows increase the likelihood of a constraint appearing. The candidate is checked by a fixed ordering, so it is expected, though not guaranteed, that fewer bad events exist in a candidate when checking a higher rank. The number of resamplings due to a lack of coverage bad event since adding the last row is used to estimate progress. To add rows readily when bad events occur early, the number of resamplings to add a row is proportional to the amount remaining to check.

3.2 Conditional Expectation Heuristic Search Algorithm

Algorithm 2, Conditional Expectation Heuristic Search (CEHS), is a greedy, one-row-at-a-time algorithm that combines ideas from conditional expectation with a heuristic to avoid constraints [2,4]. Call a credential not-yet-r-covered if it is unconstrained appearing fewer than r times or a soft constraint appearing between 0 and r times. The expectation for a row is the number of not-yet-r-covered credentials that are covered if symbols are assigned to columns randomly. Given a row with $i-1$ columns fixed to symbols and the rest free, choose a column i randomly and consider the v_i symbols to place in column i. For the symbols of that column, there is a choice of symbol that does not reduce the expectation for the row. Let $\mathcal{T}_i$ be the set of $\binom{k-1}{t-1}$ sets of t columns involving i, and $\mathcal{C}_T$ the set of possible credentials for a t-set of columns, T. Suppose column i is fixed to symbol σ. If $P(c)$ is the probability of credential c appearing and $\Lambda(c)$ is related to the coverage status of c,

$$\Lambda(c) = \begin{cases} 1 \text{ if } c \text{ covered fewer than } r \text{ times,} \\ 0 \text{ if } c \text{ covered at least } r \text{ times or } c \text{ is a soft constraint,} \end{cases}$$

define

$$value(i,\sigma) = \sum_{T\in\mathcal{T}_i}\sum_{c\in\mathcal{C}_T} \Lambda(c)P(c).$$

The expected number of not-yet-r-covered credentials newly covered by placing σ in i is $value(i,\sigma)$. The best symbol is one that maximizes $value(i,\sigma)$ without violating a hard constraint. Ties can be broken randomly.

The heuristic lies in redefining Λ. Prioritizing credentials that have been covered fewer times over those that have been covered more may be more useful than the all-or-nothing approach that works well when $\lambda = r = 1$. To avoid fixing the last symbol σ in column i of a credential that violates a hard constraint when other not-yet-r-covered credentials require σ in i, define $\Lambda(c) = -\binom{k-1}{t-1}$ for this case. There are $\binom{k-1}{t-1} - 1$ other t sets involving column i. At most, a t-set contributes 1 to $value(i,\sigma)$, so the most positive value a symbol receives from the other credentials is $\binom{k-1}{t-1} - 1$. A lookahead attempts to drive the search away from fixing symbols leading to one or more eventual hard constraints without preventing covering unconstrained credentials. The lowest benefit of placing symbol σ in column j occurs when there is one credential to be covered one remaining time with the highest number of symbols, $v = \max_{i=1}^{k}(v_i)$. The probability of being placed is lowest when all other columns in the t-set are still free, assuming j is fixed to σ. Then $P(c) = \frac{1}{v^{t-1}}$ and $\Lambda(c) = \frac{1}{r}$, so the benefit is $\frac{1}{rv^{t-1}}$. The highest cost occurs when the other t-sets involving j have $\binom{k-1}{t-1} - 1$ potential hard constraints and one free column. For each t-set, let w be the number of symbols for the free column. There are w credentials with symbols matching the $t-1$ fixed columns, and each is chosen with probability $P(c) = \frac{1}{w}$. Each t-set contributes at most $w\frac{1}{w}\Lambda$, so the total cost is $(\binom{k-1}{t-1} - 1)\Lambda$. The value of Λ must ensure that $|(\binom{k-1}{t-1} - 1)\Lambda| < \frac{1}{rv^{t-1}}$. Set $\Lambda = \frac{-1}{(\binom{k-1}{t-1}-1)ry^t}$, $y = max_{i=1}^{k}(v_i)$. When $y \geq v$,

$$\left|\left(\binom{k-1}{t-1} - 1\right)\frac{-1}{(\binom{k-1}{t-1}-1)ry^t}\right| = \left|\frac{-1}{ry^t}\right| < \left|\frac{1}{rv^{t-1}}\right|.$$

The full definition is then

$$\Lambda(c) = \begin{cases} \frac{r-\text{times } c \text{ covered}}{r} \text{ if } c \text{ is unconstrained or an appearing soft constraint,} \\ 0 \text{ if } c \text{ is a non-appearing soft constraint,} \\ \frac{-1}{(\binom{k-1}{t-1}-1)ry^t} : y = \max_{i=1}^{k} v_i, c \text{ a hard constraint with} \geq 1 \text{ free column,} \\ -\binom{k-1}{t-1}, \; c \text{ a hard constraint with 0 free columns.} \end{cases}$$

As with MTCR, a feasibility check should be conducted beforehand or an iteration limit used, as some scenarios can still result in infinite looping. CEHS lacks complete lookahead, so a series of local decisions based on the ordering of columns in an execution can lead to the placement of some hard constraint even if an anonymizing array exists. In this case, CEHS aborts and can be run again.

Algorithm 2: Conditional Expectation Heuristic Search (CEHS)

input : $r, t, k, (v_1, ..., v_k)$, and a set of constraints
output: **A** or $\emptyset$
begin
  Create an empty array, **A**, and set count of all credentials = 0
  while *some not-yet-r-covered credential remains* **do**
    Add a row to **A** with all columns free
    while *some column is free* **do**
      Randomly select a column i
      for *each symbol* $\sigma \in [v_i]$ **do**
        Compute $value(i, \sigma) = \sum_{T \in \mathcal{T}_i} \sum_{c \in \mathcal{C}_T} \Lambda(c) P(c)$
        $P(c) = \frac{\text{ways to cover c}}{\text{ways to fix free columns of T}}$
        $\Lambda(c) =$
        $\begin{cases} \frac{r - \text{count of } c}{r}, \ c \text{ unconstrained or appearing soft constraint}, \\ 0, \ c \text{ non-appearing soft constraint}, \\ \frac{-1}{(\binom{k-1}{t-1}-1) r y^t} : y = \max_{i=1}^{k} v_i, c \text{ hard constraint and } \geq 1 \text{ free column}, \\ -\binom{k-1}{t-1}, \ c \text{ hard constraint with 0 free columns}. \end{cases}$
      Place symbol σ in column i that maximizes $value(i, \sigma)$
    for *each of the credentials appearing in the row* **do**
      Update the count of the credential
      if *a hard constraint appears* **then**
        Return $\emptyset$
  Return **A**

3.3 Homogeneity Post-Optimization

We develop a post-optimization strategy in Algorithm 3 to reduce the homogeneity of an array by *crossover*, or swapping credentials between two access profiles. A first idea is to distance similar access profiles by identifying a high homogeneity access profile, u, and the access profile v with the largest $closeness(u, v)$. Then if credential c has the largest $weight(u, v, c)$, we might swap the symbols of u and access profile w in the columns of credential c for which $weight(u, w, c)$ is smallest. Computationally, this approach requires storage of the weight array whereas closeness can be computed as sums without the intermediary weights. Additionally, the view at the granularity level of weight does not inform how close u and w are on other credentials. They may be identical in all columns except some of c, and so crossover simply swaps u and w but the overall homogeneity of the array has not changed. Instead, select u and w such that $homogeneity(u)$ is highest and $closeness(u, w)$ is lowest. The key is to "decouple" u from u's group and create a link between u's group and w, an access profile outside the group, doing the same with w and w's group by swapping some credentials of u and w.

The weights give information about shared credentials so we could choose to swap any credentials c where $weight(u, w, c) = 0$. However, too many swaps results in swapping the entire row, and as u and w are chosen to have the smallest *closeness* score, they may have no credentials in common. Swapping a single credential changes up to $\binom{k}{t} - \binom{k-t}{t}$ other credentials, so how to make the best

Algorithm 3: Homogeneity Post-optimization (HP)

```
input : A, r, t, k, (v_1, ..., v_k), and a set of constraints
output: A
begin
    while generations remain do
        mostFit = A
        Compute homogeneity(i) for all rows in A
        u = max_i^N (homogeneity(i))
        for each child in the generation do
            Create a copy of A as child
            Mutate based on implementation choices
            Compute S, a set of s rows with smallest closeness(u, w), w ∈ S
            for each block of attributes based on implementation do
                Randomly select w with 1/s probability
                Swap w and u's attributes in the block
            if child is (r, t)-anonymous with lowest global homogeneity then
                Set mostFit = child
            Set A = mostFit
    Return A
```

decision without considering all possibilities is unclear. A middle path between random row resampling and computationally intensive search is to generate a set of child arrays by conducting crossover to probabilistically swap blocks of attributes between u and the S access profiles with the lowest closeness scores with u. The child with the lowest global homogeneity without violating hard constraints and meeting the anonymity guarantee becomes the parent of the next generation. As mentioned, swapping one credential changes up to $\binom{k}{t} - \binom{k-t}{t}$ other credentials in the same access profile. An affected credential that appears few times may fall below r coverage in all of the children allowed in a generation. In this case, the parent is retained and random resampling by *mutation* is conducted to allow additional appearances of the credential that is eliminated by resampling to occur elsewhere in the array to regain (r, t)-anonymity. It is not obvious how to set the mutation rate or how many and which rows to mutate. Additional tunable parameters include the set size of access profiles with which to swap, the blocksize of attributes to swap, the probability of swapping, and the number and size of generations. Stopping conditions include a generation limit, number of generations without reduced homogeneity, or meeting the expected global homogeneity.

4 Results

Comparison of MTCR and CEHS. In tests to construct $\mathsf{AA}(r, t, 10, (5^1 4^2 3^3 2^4))$, MTCR produces arrays with the same number of rows as CEHS when $t = 1$ without constraints if restricted to use the same number of rows produced by CEHS. When allowed to add additional rows, it typically adds more than needed.

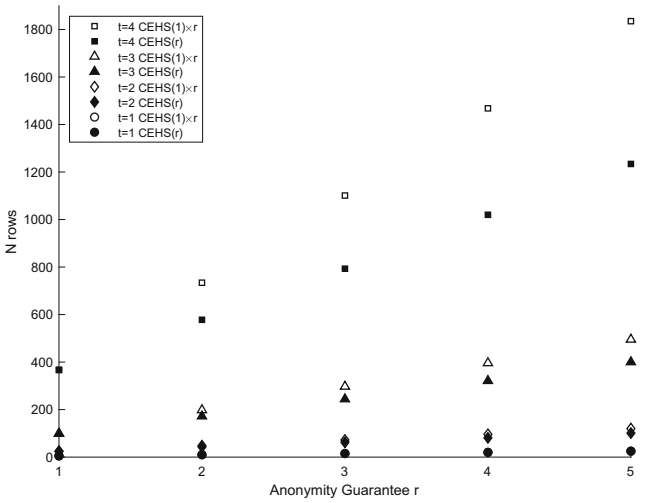

Fig. 1. CEHS versus CAcopy to build AAs with constraints

When $t = 2$ and MTCR is allowed 10^6 iterations, in general it requires more rows than CEHS to find a solution. For $t = 2$ with a hard constraint, MTCR requires about twice as many rows. For two hard constraints, MTCR does not complete in 10^6 iterations for any fixed number of rows or allowed unlimited rows. For two soft constraints, MTCR performs in fewer iterations and rows than for one hard constraint. These results suggest that randomized constructions perform poorly in the presence of hard constraints.

Comparison to Replicated Mixed-Level Covering Arrays with Constraints. A "from scratch" construction is used when attributes are assigned arbitrarily to subjects, as in key distribution. We compare the performance of CEHS against a covering array copy construction (CAcopy). CEHS is executed for $1 \leq r \leq 5$ for each $1 \leq t \leq 4$ with and 0, 6, 4, and 3 hard constraints for the values of t, respectively, to construct an $\mathsf{AA}(r, t, 10, (5^1 4^2 3^3 2^4))$. The number of rows for this construction are plotted in Fig. 1 with closed markers and labels indicating t and "CEHS(r)." To obtain an arbitrary covering array with the same constraints, CEHS is used to construct an AA with $r = 1$. Next, AAs are made for $2 \leq r \leq 5$ by stacking r copies of each covering array. The number of rows for this construction is plotted in Fig. 1 with open markers and labeled by t and "CEHS(1) $\times r$." When $t = 1$, the number of rows needed is always r times the maximum number of levels, and both constructions produce the same number of rows. For $t > 1$, the redundancy of CAcopy clearly produces more rows than CEHS. A challenge in comparing these constructions by homogeneity is that additional rows increase the likelihood that access profiles have larger credential neighborhoods. In general, an anonymizing array with more rows is less

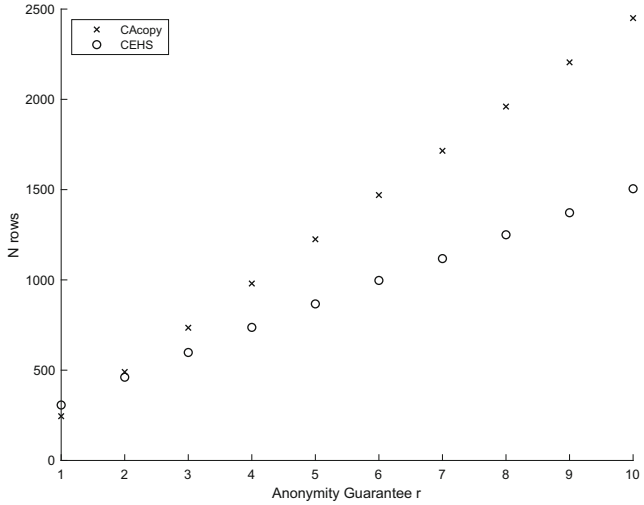

Fig. 2. CEHS versus CAcopy to build unconstrained AAs

homogeneous than one with fewer. When $t = 1$, the anonymizing arrays produced by both methods have the same number of rows for all values of r and so provide a good opportunity for comparison. In tests, the anonymizing arrays created by CEHS always have lower homogeneity scores than the copy constructed arrays. To attempt an ad-hoc comparison of the constructions in the absence of a standardized homogeneity metric that adequately compares arrays with differing numbers of rows, five rows are randomly selected from an $(r, 2)$-anonymizing array constructed by CEHS and appended to an $\mathsf{AA}(43; 2, 2, 10, (5^1 4^2 3^3 2^4))$. The rows are not constructed randomly to ensure that no hard constraints are introduced. This method is not without bias due to the pool of rows from which they are selected and is not intended for practical use. The resulting array has lower global homogeneity than the $\mathsf{AA}(48; 2, 2, 10, (5^1 4^2 3^3 2^4))$ created by CAcopy.

Comparison to Replicated Covering Arrays without Constraints. We construct a set of arrays, $\mathsf{AA}(245r; r, 3, 10, 5)$ by making $1 \leq r \leq 10$ vertical copies of a $\mathsf{CA}(245; 3, 10, 5)$ made by a conditional expectation algorithm shown to construct covering arrays with few rows efficiently [4]. As indicated in Fig. 2, when $r = 1$, the covering array has 62 fewer rows, but the CEHS algorithm produces anonymizing arrays with fewer rows for $r \geq 2$. Now, consider a row ρ in the covering array. After r copies, ρ appears (at least) r times, and this forms a cluster of rows sharing the same credentials and therefore neighborhoods. Instead, for each copy $i > 1$ and for each column j in the copy, choose a random permutation over the levels of a column, $p_{c_{i,j}} : v \mapsto v$. Each permuted copy is still a covering array, so the composed array is (r, t)-anonymous (CAperm). In this array, k independent permutations are applied to the columns of the ρth row in a copy, so the likelihood that this row closely matches ρ is reduced. In

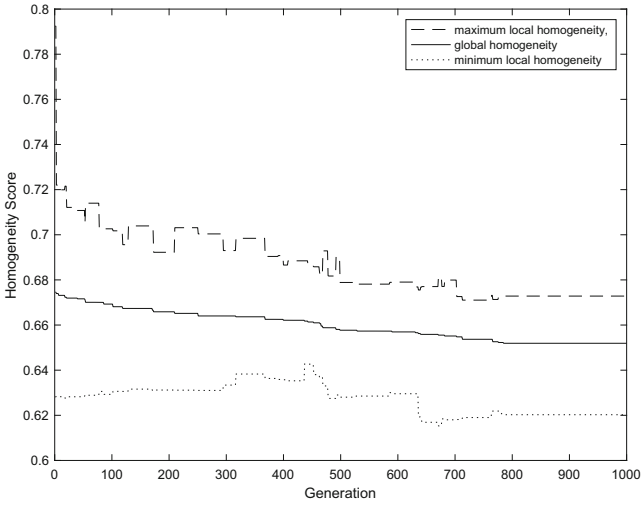

Fig. 3. HP on $\mathsf{AA}(62; 3, 2, 19, (5^1 4^2 3^3 2^4))$ with 20 children and 1000 generations

all tests, the permuted arrays have lower average and maximum homogeneity scores than by CAcopy, and in all but one data point, they have lower minimum homogeneity scores. To compare homogeneity of CAperm to CEHS, randomly generated rows are appended to the CEHS array to equalize the number of rows. For $2 \leq r \leq 10$, CEHS produces lower minimum, global, and maximum homogeneity scores than CAperm. The one exception is that CAperm produced lower maximum homogeneity for $r = 10$. This suggests that CEHS typically produces arrays with fewer rows and lower homogeneity than by copying covering arrays, even when utilizing permutations.

Evaluation of Homogeneity Post-optimization (HP). HP contains a number of tunable parameters, and details for the implementation tested here are in [7]. An example of the reduction of global homogeneity on an $\mathsf{AA}(62; 3, 2, 10, (5^1 4^2 3^3 2^2))$ generated by CEHS with six hard constraints is in Fig. 3.

5 Conclusion

Although anonymizing arrays differ from covering arrays in essential ways, constructive algorithms for covering arrays underlie useful algorithms for constructing anonymizing arrays. Indeed, this connection leads to copy constructions to produce arrays "from scratch" as well as two methods to add rows to a partial array (CEHS and MTCR). CEHS outperforms both MTCR and the copy constructions, both in terms of the number of rows generated and the homogeneity. Nevertheless, none of the construction methods examined ensures low homogeneity. To address this, we propose a "post-optimization" method (called HP) to reduce homogeneity, and provide preliminary evidence that HP is a reasonable first approach.

References

1. Bethencourt, J., Sahai, A., Waters, B.: Ciphertext-policy attribute-based encryption. In: 2007 IEEE Symposium on Security and Privacy (SP 2007), Los Alamitos, pp. 321–334. IEEE (2007)
2. Bryce, R.C., Colbourn, C.J.: A density-based greedy algorithm for higher strength covering arrays. Softw. Test. Verif. Reliab. **19**(1), 37–53 (2009)
3. Bryce, R.C., Colbourn, C.J.: Prioritized interaction testing for pair-wise coverage with seeding and constraints. Inf. Softw. Technol. **48**(10), 960–970 (2006). https://doi.org/10.1016/j.infsof.2006.03.004
4. Colbourn, C.J., Lanus, E., Sarkar, K.: Asymptotic and constructive methods for covering perfect hash families and covering arrays. Des. Codes Crypt. **86**(4), 907–937 (2017). https://doi.org/10.1007/s10623-017-0369-x
5. Hu, V.C., et al.: Guide to attribute based access control (ABAC) definition and considerations (draft). NIST Spec. Publ. **800**(162), 1–52 (2013)
6. Kapadia, A., Tsang, P.P., Smith, S.W.: Attribute-based publishing with hidden credentials and hidden policies. In: NDSS, vol. 7, pp. 179–192. Citeseer (2007)
7. Lanus, E.: Interaction testing, fault location, and anonymous attribute-based authorization. Ph.D. thesis, Arizona State University (2019)
8. Moser, R.A., Tardos, G.: A constructive proof of the general Lovász local lemma. J. ACM **57**(2), 11:1–11:15 (2010). https://doi.org/10.1145/1667053.1667060
9. Portnoi, M., Shen, C.C.: Location-enhanced authenticated key exchange. In: 2016 International Conference on Computing, Networking and Communications (ICNC), pp. 1–5. IEEE (2016)