III. FAULT DETECTION

Now consider the faults that this method can detect. Suppose that some combination of attributes exists that produces a different response than required by the policy P in k-DNF form. Tests contained in GTEST and DTEST will detect a large class of missing terms, added terms, or altered terms containing k or fewer attributes. In this section we analyze faults that will be detected, and the underlying conditions in these faults. Table 3 illustrates the fault types and detection conditions for each.

	Term	C=correct term	F=faulty term	GTEST detect condition	DTEST detect condition	notes
1	**missing**	*abc*	--	*abc*	*none*	
2	**added**	--	*ab*	*none*	*ab*	
3		*abc*	$\bar{a}b$	*none*	$\bar{a}bc$, $\bar{a}b\bar{c}$	
4		*abc*	*ab*	*none*	$ab\bar{c}$	
5		*ab*	*abc*	--	--	*no fault*
6	**altered**	*abc*	$ab\bar{c}$	*abc*	$ab\bar{c}$	
7		*abc*	*ab*	*none*	$ab\bar{c}$	
8		*abc*	$\bar{a}b$	*abc*	$\bar{a}bc$, $\bar{a}b\bar{c}$	

Table 3. Example faults and detection conditions.

k-DNF detection property: Collectively, tests from GTEST and DTEST will detect added, deleted, or altered faults with up to k attributes.

Proof outline: Three cases can be considered, for missing, added, or altered terms in the policy. The analysis for each case is keyed to the numbered examples in Table 3.

Missing term. (1) Fault detected by GTEST. If a term is missing in the faulty implementation P', then there is some combination of attributes accepted in P that is not included in P'. Since it is in P, GTEST will include the combination and the fault will be detected.

Added term. If the system incorrectly issues a *grant* response, then some combination F of attributes accepted in P' is not included in P. We consider three cases depending on the number of attributes, j, in the added term.

> $j < k$ and F is not a subset of some other term (2, 3). Detected by DTEST because the added term will be part of the non-*grant* combinations in DTEST.
>
> $j < k$ and F is a subset of some other term (4). Detected by DTEST. If the j attributes of the added term are a subset of some term C in P, then because DTEST contains all k-way combinations not excluded by R, it will include all k-way combinations of the attributes in F with settings different from C. For example, if $F = ab = 11$, and $C = abc = 110$, then DTEST will include other settings of abc, which will include $abc = 111$. But because this term includes F, it will produce an incorrect grant response, detecting the fault. Note that if P contains both abc and $ab\bar{c}$, then the result of grant for $ab = 11$ is in fact correct, since $ab\bar{c} + abc = ab$.
>
> $j < k$ and there is some term C in P that is a subset of F (5). In this case a fault does not exist because any input that produces a *grant* response from P would produce a *grant* response with F added, because F contains all attributes of C.

Altered term. Three cases can be distinguished, based on the number of attributes j in the incorrect term F as compared with k.

> $j = k$ (6). Detected by GTEST because it includes a test with the correct term, and no other combination of attributes in that test will match any other term in P.
>
> $j < k$ and F is a subset of some other term (7). Detected by DTEST if there is no other term in P' that excludes from DTEST $F \cup x$, where x is one or more attributes in C that are not in F. Example: if $C = abc$ and F is ab, then $ab\bar{c}$ is in DTEST, unless P' also contains $b\bar{c}$. Note that if DTEST includes $F \cup x$, because there is some other term D in P where $x \subseteq D$, then there is no fault because the disjunction of the altered term with the other term would not accept any attribute sets not accepted in P. Not detected by GTEST because any test that contains the attributes of the correct term C will contain all attributes of a subset of C.
>
> $j < k$ and F is not a subset of altered term C (8). Detected by GTEST because it will include C, and P' will not match C. □

If more than k attributes are included in the altered term, some faults are still detected.

> $j > k$ and C is not a subset of F. Detected by GTEST because C will be included in GTEST but will produce a deny response.
>
> $j > k$ and C is a subset of F. Not detected by DTEST because DTEST excludes C, and therefore excludes F because it contains C. Not necessarily detected by GTEST because the settings of attributes x in F but not C may result in $C \cup x = F$. This case can be resolved by strengthening the covering array DTEST, using an array of strength $k+i$ to detect faulty terms with up to i additional attributes.

IV. TRADEOFFS AND PRACTICAL CONSIDERATIONS

The process scales easily to systems with a large number of attributes that must be included in access decisions. Because the number of rows in a covering array grows only with log n for n variables/attributes at a given number of attributes

and values, a larger policy specification, involving many more attributes, requires only a few additional tests. For example, it is possible to cover all 3-way combinations of 100 boolean variables with 45 tests, increasing only to 57 tests for 300 variables.

The most significant limitation for this approach occurs where terms in access control rules contain a large number of attribute values per attribute. Although covering array size grows only with log n, the value of k for the k-DNF form of rules is an exponent in the number of combinations that must be covered, and consequently the number of rows increases with v^k, for v attribute values. If terms in the rules contain more than six or seven attributes, it may not be practical to generate covering arrays, given the limitations of today's algorithms. However, a large number of tests is not a barrier, because the structure of the solution resolves the oracle problem by ensuring that every test in GTEST should produce a response of *grant* and every test in DTEST should produce a response of *deny*. This means that tests can be fully automated, making it possible to execute a large number of tests.

Table 5 shows test set sizes for a variety of configurations, assuming a value of $p = 4$ terms per rule. The column N tests gives the size N of a covering array for k-way combinations of n attributes with v values each. The size of DTEST is $m \times N$, since there will be one covering array per rule tested. Note that the test time for even a large test set of more than 600,000 tests would be roughly 10 minutes, assuming a time of 1 ms per test. In addition, since the tests are independent, testing can be divided among as many processors as are available, so even millions of tests could be tractable.

k	*v*	*n*	*m*	N tests	#GTEST	#DTEST
3	2	50	20	36	80	720
			50		200	1800
		100	20	45	80	900
			50		200	2250
	4	50	20	306	80	6120
			50		200	15300
		100	20	378	80	7560
			50		200	18900
	6	50	20	1041	80	20820
			50		200	52050
		100	20	1298	80	25960
			50		200	64900
4	2	50	20	98	80	1960
			50		200	4900
		100	20	125	80	2500
			50		200	6250
	4	50	20	1821	80	36420
			50		200	91050
		100	20	2337	80	46740
			50		200	116850
	6	50	20	9393	80	187860
			50		200	469650
		100	20	12085	80	241700
			50		200	604250

Table 4. Test set sizes.

V. HIPAA PRIVACY EXAMPLE

The following text is an excerpt from Health Insurance Portability and Accountability Act (HIPAA) rules that specify when a health care organization must treat a patient's personal representative of as the individual [10]. (This policy fragment has been used in previous work on formal specification of natural language policies [11].) Typical cases include a parent making decisions on behalf of a child.

> *(g)(1) Standard: Personal representatives. As specified in this paragraph, a covered entity must, except as provided in paragraphs (g)(3) and (g)(5) of this section, treat a personal representative as the individual for purposes of this subchapter.*
>
> *(2) Implementation specification: adults and emancipated minors. If under applicable law a person has authority to act on behalf of an individual who is an adult or an emancipated minor in making decisions related to health care, a covered entity must treat such person as a personal representative under this subchapter, with respect to protected health information relevant to such personal representation.*
>
> *(3)(i) Implementation specification: unemancipated minors. If under applicable law a parent, guardian, or other person acting in loco parentis has authority to act on behalf of an individual who is an unemancipated minor in making decisions related to health care, a covered entity must treat such person as a personal representative under this subchapter, with respect to protected health information relevant to such personal representation, except that such person may not be a personal representative of an unemancipated minor, and the minor has the authority to act as an individual, with respect to protected health information pertaining to a health care service, if:*
>
> *(A) The minor consents to such health care service; no other consent to such health care service is required by law, regardless of whether the consent of another person has also been obtained; and the minor has not requested that such person be treated as the personal representative; (B) The minor may lawfully obtain such health care service without the consent of a parent, guardian, or other person acting in loco parentis, and the minor, a court, or another person authorized by law consents to such health care service; or (C) A parent, guardian, or other person acting in loco parentis assents to an agreement of confidentiality between a covered health care provider and the minor with respect to such health care service.*

Step 1: Review the text to identify attributes or variables that must be considered in access rules. Attributes in statutes may represent existence of various documents signed by the parties, among other basic attributes such as age, citizenship, etc. For this example, we consider the rules specified in (g)(3)(i)(A), for cases in which a minor has the authority to act as an individual. Each attribute is given a short mnemonic name in the covering array. These are shown below by bracketing each attribute, with a variable name annotation:

(A) The **{minor consents : mc}** to such health care service**;** no **{other consent : oc}** to such health care service is required by law, regardless of whether the consent of another person has also been obtained; **and** the minor has not **{requested that such person be treated as the personal representative : mr}**; (B) The **{minor may lawfully obtain : lo}** such health care service without the consent of a parent, guardian, or other person acting in loco parentis, **and** the **{minor : mc}**, a **{court : cc}**, or **{another person : oc}** authorized by law consents to such health care service; **or** (C) A **{parent, guardian, or other person acting in loco parentis assents to an agreement of confidentiality : pc}** between a covered health care provider and the minor with respect to such health care service.

Step 2: Convert the text description to rules in *k*-DNF form, in this case, 3-DNF. This mapping is as shown in Table 5. Note that the "or" connector prior to clause (C) indicates a disjunction of the three clauses.

Text	Attributes
(A) The **{minor consents : mc}** to such health care service**;** no **{other consent : oc}** to such health care service is required by law, regardless of whether the consent of another person has also been obtained; **and** the minor has not **{requested that such person : mr}** be treated as the personal representative;	expression: mc && ~oc && ~mr attribute sets: {mc, ~oc, ~mr}
(B) The **{minor may lawfully obtain : lo}** such health care service without the consent of a parent, guardian, or other person acting in loco parentis, **and** the **{minor : mc}**, a **{court : cc}**, or **{another person : oc}** authorized by law consents to such health care service;	expression: lo && (mc||cc||oc) = lo && mc || lo && c || lo && oc attribute sets: {lo, mc}, {lo, cc}, {lo, oc}
(C) A **{parent, guardian, or other person acting in loco parentis assents to an agreement of confidentiality : pc}**	expression: pc attribute sets: {pc}

Table 5. Mapping of text to attributes.

Step 3: Determine the maximum number of *AND* connectives in access rule conditions. In the HIPAA example, three attributes are conjoined in Sect. (g)(3)(i)(A): "*minor consents …; no other consent to such health care service is required by law, …; and the minor has not requested that such person be treated as the personal representative"*. Because the expression above is in 3-DNF, with a maximum of three attributes in conjunction, a 3-way covering array is sufficient to consider all relevant combinations of attribute values. Note that other practical cases may involve more than 3-way combinations of attributes in terms, but the process would be the same as in this illustrative example. Combining these terms, we have:

mc && ~oc && ~mr || lo && (mc || cc || oc) || pc → *grant*
=
mc && ~oc && ~mr || lo&&mc || lo&&cc || lo&&oc || pc → *grant*

Step 4:
GTEST: Generate tests for GTEST, with one test for each term and other terms false. Tests are shown in Figure 2.

	mc	oc	mr	lo	cc	pc
1	1	0	0	0	0	0
2	1	0	1	1	0	0
3	0	1	0	1	0	0
4	0	0	0	1	1	0
5	0	0	0	0	0	1

GTEST

	mc	oc	mr	lo	cc	pc
1	0	0	0	0	0	0
2	0	0	1	1	0	0
3	0	1	0	0	1	0
4	0	1	1	0	0	0
5	1	0	1	0	1	0
6	1	1	0	0	0	0
7	1	1	1	0	1	0
8	0	0	0	1	0	0
9	0	0	0	0	1	0
10	1	0	1	0	0	0
11	1	1	0	0	1	0
12	0	1	1	0	1	0

DTEST

Figure 2. Completed GTEST and DTEST arrays.

DTEST: Compute a covering array for DTEST for the value of *t* determined in Step 2, using ~*R* as a constraint. For illustration purposes, we include a covering array of the six variables without constraints in Figure 3, followed by a covering array computed with the expression above as a constraint. Note that no terms from the expression above can be found in the constrained array. For example, because array (1) includes all 3-way combinations, mc && ~oc && ~mr is present, but in (2) it is not found, although all other 3-way combinations of these variables are present in the array (2).

	mc	oc	mr	lo	cc	pc
1	0	0	0	0	0	0
2	0	0	1	1	1	1
3	0	1	0	1	0	1
4	0	1	1	0	1	0
5	1	0	0	1	1	0
6	1	0	1	0	0	1
7	1	1	0	0	1	1
8	1	1	1	1	0	0
9	1	1	0	0	0	0
10	0	0	1	1	0	0
11	0	0	0	0	1	1
12	1	1	1	1	1	1

(1) Covering array without constraint

	mc	oc	mr	lo	cc	pc
1	0	0	0	0	0	0
2	0	0	1	1	0	0
3	0	1	0	0	1	0
4	0	1	1	0	0	0
5	1	0	1	0	1	0
6	1	1	0	0	0	0
7	1	1	1	0	1	0
8	0	0	0	1	0	0
9	0	0	0	0	1	0
10	1	0	1	0	0	0
11	1	1	0	0	1	0
12	0	1	1	0	1	0

(2) Covering array with constraint

Figure 3. DTEST array compared with unconstrained array.

Security and Reliability-Companion (SERE-C), 2014 IEEE Eighth International Conference on (pp. 41-49). IEEE.

[17] Martin, E. (2006, October). Automated test generation for access control policies. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications* (pp. 752-753). ACM.

[18] Martin, E., & Xie, T. (2007, May). Automated test generation for access control policies via change-impact analysis. In *Proceedings of the Third International Workshop on Software Engineering for Secure Systems* (p. 5). IEEE Computer Society.

[19] Fisler, K., Krishnamurthi, S., Meyerovich, L. A., & Tschantz, M. C. (2005, May). Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th international conference on Software engineering* (pp. 196-205). ACM.

[20] V. Hu, D.R. Kuhn, T. Xie, Property Verification for Generic Access Control Models, IEEE/IFIP International Symposium on Trust, Security, and Privacy for Pervasive Applications, Shanghai, China, Dec. 17-20, 2008.

[21] ACPT Home Page, http://csrc.nist.gov/groups/SNS/acpt/access_control_policy_testing.html

[22] Bertolino, A., Lonetti, F., & Marchetti, E. (2010, September). Systematic XACML request generation for testing purposes. In *Software Engineering and Advanced Applications (SEAA), 2010 36th EUROMICRO Conference on* (pp. 3-11). IEEE.

[23] Bertolino, A., Daoudagh, S., Lonetti, F., & Marchetti, E. (2012, April). Automatic XACML requests generation for policy testing. In *Software Testing, Verification and Validation (ICST), 2012 IEEE Fifth International Conference on* (pp. 842-849). IEEE.

[24] D. R. Kuhn, V. Okun, Pseudo-exhaustive Testing For Software, 30th NASA/IEEE Software Engineering Workshop, April 25-27, 2006

[25] Working DRAFT Information technology - Next Generation Access Control –Generic Operations and Data Structures (NGAC-GOADS)), INCITS 499-2013, American National Standard for Information Technology, American National Standards Institute, April 2014.